

EINDHOVEN UNIVERSITY OF TECHNOLOGY  
Department of Mathematics and Computer Science

**MASTER'S THESIS**

**Mapping  
an LTE Baseband Receiver  
on a Multi-Core Architecture**

**by  
J.P.T. Dobbelsteen**

Document Type:	Master thesis Eindhoven University of Technology (TU/e)
MSc work:	Performed at ST-Ericsson, Technology and Tools, Advanced R&D, Eindhoven
Period of work:	Januari 17 <sup>th</sup> , 2009 – November 2 <sup>nd</sup> , 2009
Supervisor:	prof. dr. C.H. van Berkel
Tutor:	O. Paker PhD
Comittee:	dr. Ir. B. Mesman (TU/e) R.J.M. Nas MSc



# Mapping an LTE Baseband Receiver on a Multi-Core Architecture

Master Thesis

## Abstract

New standards for mobile phones have arrived and these must be analyzed in order to make integrated hardware the first time right. Creating the hardware platform and software on a semiconductor is prohibitively expensive and time-consuming. Rather, the method is intended to explore the design before the real implementation is created.

Mobile phone users desire to access to the Internet, allowing the searching of information, communicating with others using e-mail, but also chatting, listening to Internet radio and watching movies and television from any location in the world. The new cellular standards, like LTE and LTE-Advanced, provide the higher data rates to make these applications possible.

A multi-core is required to provide the required computational power for implementing the new standards. Software has been developed that simulates multiprocessor hardware. It allows different software models to be mapped onto the hardware. This will be done during compile-time or run-time. The goal is to analyze the energy consumption for different mappings and hardware architectures.

# Legal Information

© Copyright ST-Ericsson, 2009. All Rights Reserved.

## Disclaimer

The contents of this document are subject to change without prior notice. ST-Ericsson makes no representation or warranty of any nature whatsoever (neither expressed nor implied) with respect to the matters addressed in this document, including but not limited to warranties of merchantability or fitness for a particular purpose, interpretability or interoperability or, against infringement of third party intellectual property rights, and in no event shall ST-Ericsson be liable to any party for any direct, indirect, incidental and or consequential damages and or loss whatsoever (including but not limited to monetary losses or loss of data), that might arise from the use of this document or the information in it.

ST-Ericsson and the ST-Ericsson logo are trademarks of the ST-Ericsson group of companies or used under a license from STMicroelectronics NV or Telefonaktiebolaget LM Ericsson.

All other names are the property of their respective owners.

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Overview of LTE Cellular Standard	10
1.1.1	Orthogonal Frequency-Division Multiplexing	11
1.1.2	MIMO	12
1.1.3	Reference Symbols	13
1.1.4	Modes of operation	15
1.2	Overview of Baseband Processing	16
1.2.1	Digital Front-End	17
1.2.2	Modem (Inner Receiver)	17
1.2.3	Codec (Outer Receiver)	17
1.3	Data Flow Model	18
1.3.1	Synchronous Data Flow	18
1.3.2	Cyclo-Static Data Flow	20
1.3.3	Scheduling	20
1.3.4	Compact Representation	22
<b>2</b>	<b>Problem Statement</b>	<b>23</b>
2.1	Goals	23
2.2	The Approach	24
<b>3</b>	<b>Application Modelling</b>	<b>28</b>
3.1	Task Graph Flow	28
3.2	Task Model	29
3.2.1	Conversion to Finite FIFO Buffers	29
3.2.2	SDF Task Implementation	30
3.2.3	CSDF Task Implementation	31
3.2.4	FIFO Channel Implementation	32
3.3	Long Term Evolution Task Graph	33
3.3.1	Inner receiver overview	33
3.3.2	Dynamism of the application	34
3.3.3	Antenna Pre-processor	35
3.3.4	Antenna combiner	37
<b>4</b>	<b>Hardware Modelling</b>	<b>39</b>
4.1	Hardware Composition	39
4.2	Hardware Flow	40
4.3	Processing Units	41

4.4	Interconnect and Memory	42
4.4.1	Local memory	43
4.4.2	Shared bus	43
4.4.3	Uniform Access Memory	46
4.4.4	Memory Caches	49
4.5	Software facilities	50
4.5.1	Task manager	50
4.5.2	System Controller	50
<b>5</b>	<b>Energy reporting</b>	<b>54</b>
5.1	Energy model	55
5.2	Dynamic Frequency and Voltage Scaling	55
5.3	Processing Unit Power	56
5.4	Interconnect and Memory Power	57
<b>6</b>	<b>Mapping</b>	<b>58</b>
6.1	Mapping Overview	58
6.1.1	Processor assignment	59
6.1.2	Memory assignment	59
6.2	Classification for scheduling algorithms	61
6.3	Static mapping	61
6.4	Dynamic mapping	62
6.5	Communication-optimized mapping	64
<b>7</b>	<b>Results and Discussion</b>	<b>68</b>
7.1	Experiment Set-up	68
7.2	Experiment 1: Simulation Validation	70
7.3	Experiment 2: Application Dynamism	71
7.3.1	Bandwidth scaling	71
7.3.2	Energy optimization	73
7.4	Experiment 3: Dynamic assignment with Clustering	76
7.5	Experiment 4: LTE Multi Processor Scaling	78
7.5.1	Dynamic mapping	79
7.5.2	Dynamic mapping with clustering	81
7.6	Discussion	81
<b>8</b>	<b>Conclusions and Future Work</b>	<b>83</b>
8.1	Conclusions	83
8.2	Future work	83

<b>9</b>	<b>References</b>	<b>84</b>
<b>10</b>	<b>Acronyms and Terms</b>	<b>86</b>

## Preface

This Master's thesis is the result of my graduation project at the end of the Embedded Systems program of the Eindhoven University of Technology. The project was started in January 2009 and finished in November 2009. The work has been performed in the Advanced Research and Development group at ST-Ericsson in Eindhoven.

I would like to thank Prof. Kees van Berkel for his guidance and being the supervisor during this project. I like to thank Özgün Paker for his guidance and his help with understanding of the LTE cellular standard and multiprocessor systems. I like to thank Rick Nas for his support and help with the various tools. Also I would like to thank Orlando Moreira for his help with explaining scheduling and data flow graphs. I greatly appreciate the support from my family, girlfriend and friends.

- Joris Dobbelsteen

# 1 Introduction

Today people want to remain connected wherever they go. The first mobile phones have enabled people to make and receive phone calls from any location at any time. Now, there is a desire to access to the Internet, allowing the searching of information, communicating with others using e-mail, but also chatting, listening to Internet radio and watching movies and television from any location in the world. As a result, users demand even higher data rates to make these applications possible.

The evolution of cellular communication standards reflects these trends. Where Global System for Mobile communication (GSM) in 1990 allowed for voice only, High Speed Download Packet Access (HSDPA) enabled communication with the Internet at speeds up to 14 Mbps of communication, the upcoming UMTS Long Term Evolution (LTE) cellular standard allows for 300 Mbps. Within a couple years it will be followed by LTE-Advanced, enabling downloads with speeds of up to 1 Gbps. However, these protocols that allow for faster communication come at a cost. A higher data rate implies an increase of not only the amount of data needed to be received or transmitted, but also an increase in the complexity of the algorithms themselves. As such, the mobile phone has to execute more complex algorithms at a faster rate. It is expected that LTE-Advanced will require at least 10 times the computational power of LTE, since it uses 5 times the bandwidth and double the number of antennas.

The upcoming LTE-Advanced cellular communication standard is set to arrive at 2015. At this time, the semiconductor manufacturing technology has been improved and the 25 nm technology nodes will become available. This technology, compared to current 45 nm, will allow up to 4 times as many transistors on the same area. At the same time, they allow higher clock speeds and lower power consumption. However these manufacturing advancements alone will not be able to bridge the 10x gap.

In addition, the phones that have all these features are no longer limited to just the cellular network. We want to share pictures with others and attach headphones wireless. This demands communication directly with other devices or using a wireless LAN. A new high-end phone capable of supporting LTE would still need to support the older cellular standards like GSM, EDGE, UMTS, HSDPA in case LTE is not available. It also needs to support different wireless protocols, such as IEEE 802.11b, IEEE 802.11g and 802.11n for wireless LAN, Bluetooth and FM radio. The different standards require different algorithms. This requires increasingly more flexible hardware that can support these standards.

With more information to be processed, the mobile phone still has to be powered by the same battery, which provides the same amount of energy, with good standby and operating times. Although battery capacity increases with time, the advancements are slow and do not keep up with the demand placed on them. Rather, the system needs to be designed to maintain low power consumption for the typical situation, comparable to mobile phones which are currently available on the market.

In order to meet the demand for high performance, increased flexibility and low power, research is needed for a hardware architecture which is capable of supporting the

upcoming cellular standards. The use of multiprocessor will be inevitable to meet the performance demands [5], but it will require new technologies to distribute the application over the hardware. Even though nearly all modern personal computers are multiprocessors, the challenges on an embedded platform are completely different. Since a personal computer is intended to run many different applications, it is tuned towards ease of programming. Since an embedded system is highly tuned for a very specific problem domain, it contains a careful mix of dedicated hardware, programmable DSPs and generic processors. The components must be picked in order to balance the specific requirements for the application it is intended to run.

Next, power saving technologies like dynamic voltage and frequency scaling must be investigated. It might be advantageous to use multiple slower, but more energy efficient processors. However, as more processors will be executing a single application, communication will need to increase to communicate data and synchronize the state. So while the processors will be consuming less energy, the communication may offset this advantage. With increasingly complex systems, the trade-offs become less obvious.

For new cellular chipset, ST Ericsson employs the Embedded Vector Processor (EVP). This is a fully programmable vector processor specifically tuned for algorithms required in cellular standards. It provides a fully programmable platform, where software, rather than hardware, defines the radio algorithms which are being used [4]. This concept, called Software-Defined Radio (SDR) is being actively researched by ST Ericsson. It allows for a single chip to be used for multiple standards and can even run multiple wireless standards at the same time.

In this thesis, various hardware architectures are explored and application is presented. The hardware architecture is intended to be capable of running LTE, therefore an introduction into the LTE cellular standard is given in Section 1.1 and an introduction in the operation of a radio receiver is given in Section 1.2. To model the application, the synchronous data flow and cyclo-static data flow model will be used, which are introduced in Section 1.3. The problem statement and the approach for this master thesis are given in Chapter 2.

## 1.1 Overview of LTE Cellular Standard

This thesis focuses on analyzing hardware architectures capable of supporting the Long Term Evolution (LTE) cellular standard. In this section the standard is briefly explained, as it helps understanding the application model used for this study. LTE is described in [23], [11] and [6].

The evolution of the mobile phone has started with GSM, enabling people to have phone conversations at any locations. Later standards enabled data services, such as Internet access and watching television. These features require increasingly higher data rates; however the bandwidth assigned by governments to mobile phone operations is still limited. In Table 1 the data rates of the cellular standards are given<sup>1</sup>. The newer standards provide a better spectral efficiency, allowing more data to be transmitted using the same

---

<sup>1</sup> 3GPP Releases. <http://www.3gpp.org/Releases> (accessed October 2009)

bandwidth. However the cost comes in increased complexity, requiring more processing by the transmitters and receivers.

LTE is the latest evolution for cellular standards and will soon be followed by LTE-Advanced. Both LTE and LTE-Advanced are based on Orthogonal Frequency Division Multiplexing (OFDM) modulation scheme. Both standards employ multi-antenna technology, Multiple Input Multiple Output (MIMO), to provide a significant increase in data rates without requiring additional bandwidth or transmit power.

Standard	Introduction	Data rate	Modulation scheme
GSM	1990	9.6 kbps	TDMA / FDMA
GPRS	1997	57.6 kbps	
EDGE	2003	236 kbps	
UMTS	1999	386 kbps	CDMA
HSDPA	2002	14 Mbps	
HSPA+ R7	2007	42 Mbps	
HSPA+ R8	2008	42 Mbps	
LTE	2008	300 Mbps	OFDM
LTE-Advanced	2015	> 1 Gbps	

Table 1 Overview of 3GPP cellular standards

LTE uses Orthogonal Frequency Division Multiplexing (OFDM) modulation, which is explained in Section 1.1.1. The use of multiple antennas and the wireless channel model is explained in Section 1.1.2. The reference symbols (RS) which are used for channel estimation are explained in Section 1.1.3. Finally, the various different modes supported by LTE are described in Section 1.1.4.

### 1.1.1 Orthogonal Frequency-Division Multiplexing

Orthogonal Frequency-Division Multiplexing (OFDM) utilizes a large number of carriers located on closely-spaced frequencies, as depicted in Figure 1. Each carrier is modulated using a more conventional modulation scheme, such as phase shift keying (PSK) or quadrature amplitude modulation (QAM). Even though the symbol rate is low for each carrier, the large number of carriers allows for high data rates. For LTE, the number of carriers ranges between 72 and 1200.

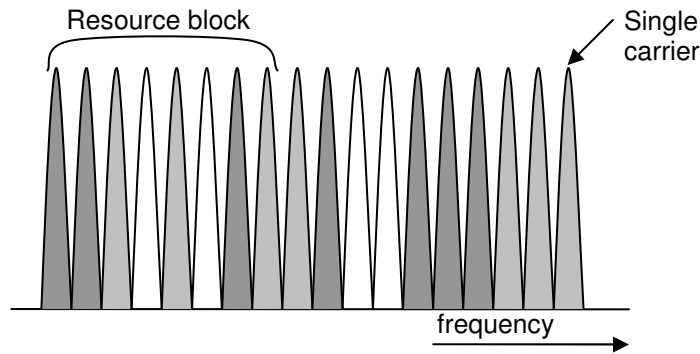


Figure 1 OFDM carrier

In the frequency domain, OFDM utilizes a large number of carriers. All OFDM carriers combined which are transmitted at a single time are called an OFDM symbol, as shown in Figure 2. A single OFDM carrier in an OFDM symbol is called a resource element. A group of 7 OFDM symbols is called a slot. Two slots, or 14 OFDM symbols, are called a sub frame. Every sub frame is transmitted in 1 millisecond. A group of 12 carriers by 14 symbols is called a resource block.

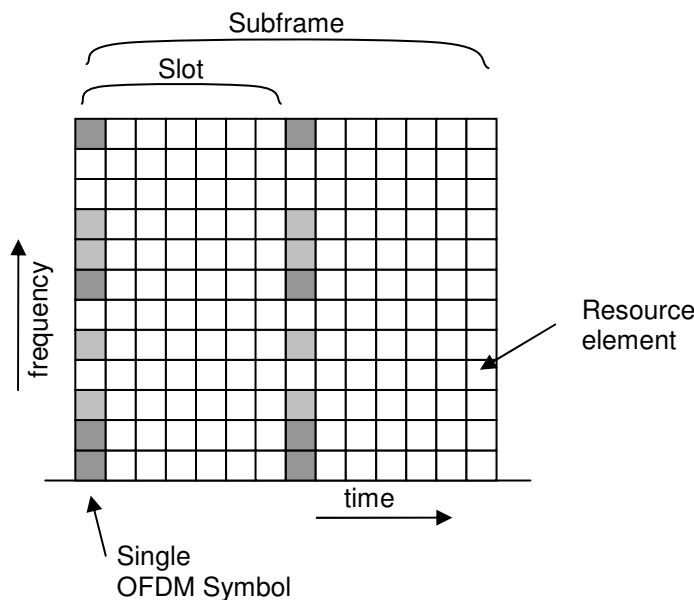


Figure 2 Single LTE Resource Block

### 1.1.2 MIMO

For cellular standards, the typical communication is between a base station and a mobile phone. In Figure 3, a simple wireless channel is depicted, where the signal  $x$ , transmitted by the base station, passes through a radio channel  $H$  and suffers noise  $n$  before being received by the mobile phone. The mobile phone receive signal  $y$ . The relation between the received and transmitted signal is mathematically expressed as  $y = H(x) + n$ . By estimating the transfer function  $H$  the effects of the channel can be compensated by the receiver.

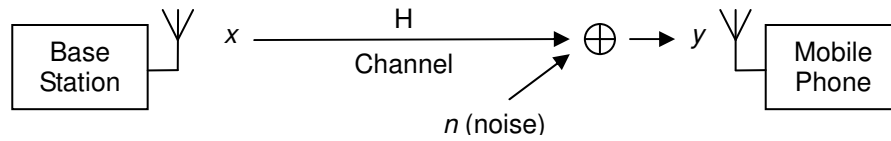


Figure 3 Wireless channel

The MIMO mode is described as  $M \times N$ , where  $M$  describes the number of transmit antennas and  $N$  the number of receive antennas. The wireless channel can now be described by the following generic formula:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} H_{0,0} & H_{1,0} & \cdots & H_{N,0} \\ H_{0,1} & H_{1,1} & & H_{N,1} \\ \vdots & & \ddots & \vdots \\ H_{0,M} & & \cdots & H_{N,M} \end{bmatrix} \circ \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} n_0 \\ n_1 \\ \vdots \\ n_M \end{bmatrix}$$

The MIMO technique called spatial multiplexing allows multiple independent streams of data are transmitted. The so-called spatial multiplexing order is  $s = \min(M, N)$ . It means  $s$  independent streams can be transmitted, thereby improving the spectral efficiency by factor  $s$ .

In Figure 4, the regular situation with 4x2 MIMO is shown.

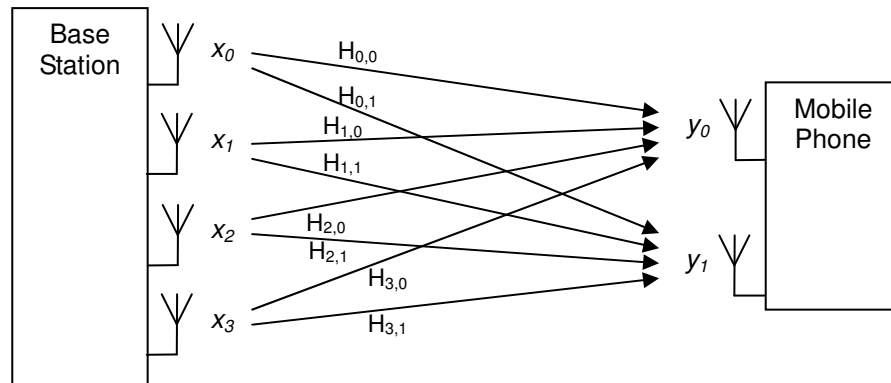


Figure 4 4x2 MIMO wireless channels

### 1.1.3 Reference Symbols

The LTE receivers estimate the channel by relying on knowing what the transmitter will send. Therefore, the transmitter periodically transmits signals which are known beforehand by the receiver; they are called reference symbols (RS) in the LTE standard. In this section it is explained how the reference symbols can be used to estimate the channels.

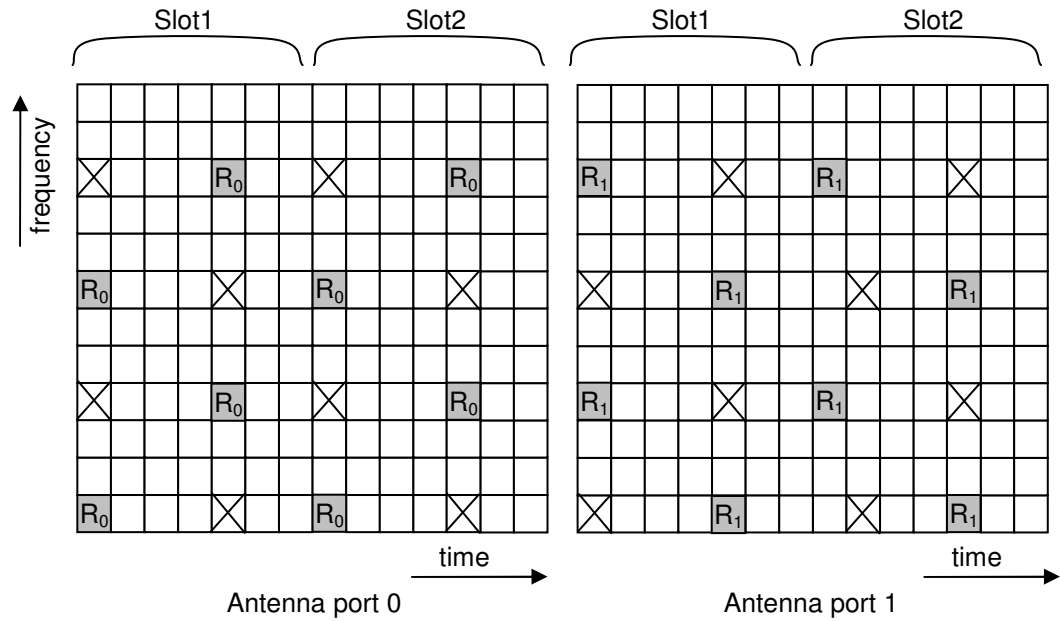


Figure 5 Reference Symbols for 2 transmitter antennas in single resource block on different ports

The reference symbols are transmitted at predefined OFDM symbols and carriers. The reference symbols are transmitted on different symbols and carriers for each antenna, at which point the other antennas will not be transmitting anything. This is shown in Figure 5, where  $R_x$  indicates that a reference symbol is transmitted by antenna  $x$ . The crosses indicate that no signal is transmitted by that antenna. For reference symbols from the first antenna, the formula simplifies to:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} H_{0,0} & H_{1,0} & \cdots & H_{N,0} \\ H_{0,1} & H_{1,1} & & H_{N,1} \\ \vdots & & \ddots & \vdots \\ H_{0,M} & & \cdots & H_{N,M} \end{bmatrix} \circ \begin{bmatrix} x_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} n_0 \\ n_1 \\ \vdots \\ n_M \end{bmatrix} = \begin{bmatrix} H_{0,0}(x_0) + n_0 \\ H_{0,1}(x_0) + n_1 \\ \vdots \\ H_{0,M}(x_0) + n_M \end{bmatrix}$$

In this case  $x_0$  is the reference symbol and is known beforehand. When the noise component can be estimated or is small enough to be ignored, it becomes possible to determinate the transfer functions  $H_{0,m}$  for each channel from transmit antenna 0 to receive antenna  $m$ .

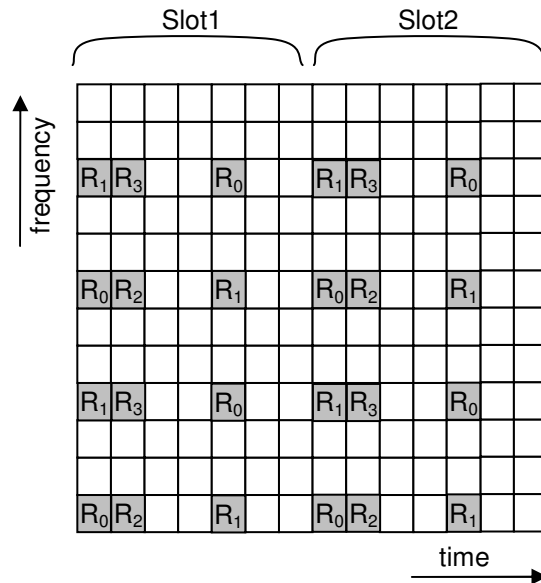


Figure 6 Reference Symbols for 4 transmitter antennas in single resource block

For LTE, at most 4 transmit antennas are supported at the base station. Hence, 4 sets of reference symbols are transmitted. In Figure 6, the reference symbols which are transmitted are combined into a single view, as they are received. The  $R_x$  is a reference symbol, which is transmitted by antenna  $x$ . Other antennas will not transmit a signal at this resource element.

### 1.1.4 Modes of operation

In [2] the physical layer requirements are detailed. In the cellular infrastructure, the area to be covered will be split into a number of cells. A cell is a geographical region which can be covered by a single base station, which can have a radius of up to 100 km. The LTE cellular standard provides a wide variety of modes, so each cell can be adjusted according to the number of users in each cell, the geographical size of cell and government regulations. It is expected that at least 200 users in active state can be supported in spectrums up to 5 MHz, while a much higher number of total users are supported in a single cell.

The different bandwidths specified by LTE are shown in Table 2. The supported MIMO modes are shown in Table 3. The mobile phone is required to have at least 2 receive antennas, and the base station is required to have at least 2 transmit antennas.

Bandwidth (MHz)	Downlink (Mbps)	OFDM carriers
1.4	18	72
3.0	36	144
5	75	300
10	150	600
15	225	900
20	300	1200

Table 2 Bandwidths specified for LTE

Transmit antennas	Receive Antennas	
	2	4
2	2x2 (2 spacial streams)	2x4 (2 spacial streams)
4	4x2 (2 spacial streams)	4x4 (4 spacial streams)

Table 3 MIMO modes specified for LTE

## 1.2 Overview of Baseband Processing

The transmitter and receivers are divided into several parts that handle different stages in the processing of the signals [4], [11]. In this thesis, only the modem part of the receiver will be considered.

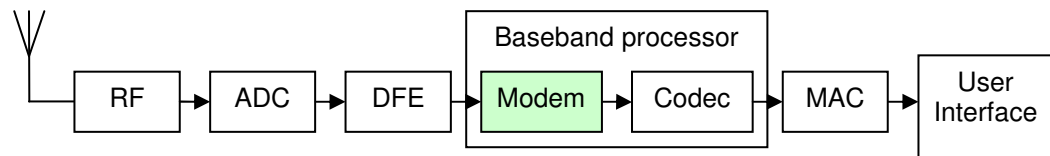


Figure 7 Radio Receiver

The antenna will receive a signal, which will be down-converted by the Radio Frequency (RF) unit from a high frequency to an intermediate frequency. The analogue to digital converter (ADC) will convert the signal to the digital domain. The digital front-end (DFE) will perform filtering of the digital signal to prepare it for processing by the baseband. The baseband is responsible for processing the signal according to the required wireless standard, such as LTE. The media access controller (MAC) manages addressing of mobile phones and channel access. Eventually the processed signal reaches the user. The DFE and baseband will be detailed in the following sections.

## 1.2.1 Digital Front-End

The Digital Front-End is responsible for filtering the received signal before further processing by the modem. The typical operations include DC-offset, frequency offset and sample rate corrections.

The algorithms executed by the DFE are typically very similar between different standards. Therefore the DFE is best implemented in dedicated hardware, where parameters of the filters can be configured [4].

## 1.2.2 Modem (Inner Receiver)

The modem, also called inner receiver, is the part that handles, amongst others, the demodulation, synchronization, estimation of the channel and equalization of the MIMO stream. The modem is the main differentiator for baseband implementations, since:

- The various standards that a mobile phone has to support are highly diverse, since they might be using fundamentally different modulation schemes.
- Programmable hardware is capable of supporting evolving standards. Changes to the standard can be incorporated at a later time. This allows draft standards to be supported early on and future enhancements to the standard to be incorporated.
- There is a high computational load, especially for new standards, such as LTE and LTE-Advanced. The right mix between performance, flexibility and power efficiency is extremely challenging.
- The wireless standards can be implemented in different ways. These different implementations can trade receive and transmit performance against the cost of hardware required to run the algorithm. These allow differentiation in the market.

## 1.2.3 Codec (Outer Receiver)

The codec, also called outer receiver, handles error correction. As noise and interference can cause distortion of the received signal, the wireless standards will require transmitters to add redundant information. This improves the reliability of the received signal, since it allows the receiver to later correct the incorrectly received data.

The algorithms used by the standards are not highly diverse. Typically Turbo, Viterbi and Reed-Solomon error correction coding is being used, commonly combined with rearrangement of the data to improve the performance of the coding scheme. Since only a limited number of functions have to be supported to support multiple standards, dedicated hardware with limited flexibility is sufficient.

## 1.3 Data Flow Model

The algorithms in the modem which are required for implementing the LTE standard have to be modelled. By using data flow, a high-level description of this algorithm can be implemented and analyzed.

In data flow, the application model describes how data is transmitted through a system. The model contains actors, which can perform operations on the data they receive. In additions, there are directed edges, or channels, which describe between which two actors data is transferred. The Synchronous Data Flow (SDF) is explained in Section 1.3.1. The Cyclo-Static Data Flow (CSDF) model is explained in 1.3.2.

### 1.3.1 Synchronous Data Flow

The SDF model, introduced in [14], also fixes the amount of data which is produced by every actor. In Figure 8 an SDF graph with 4 actors<sup>2</sup> - A, B, C and D - is shown. The connections between the actors are called edges. An actor is allowed to fire, i.e. perform its operation, when all incoming edges contain sufficient tokens. After execution, tokens will be produced on the outgoing edge. For example, actor A can fire when at least 1 token is available on the edge from D to A. After firing, it will put 1 token on the edge from A to B.

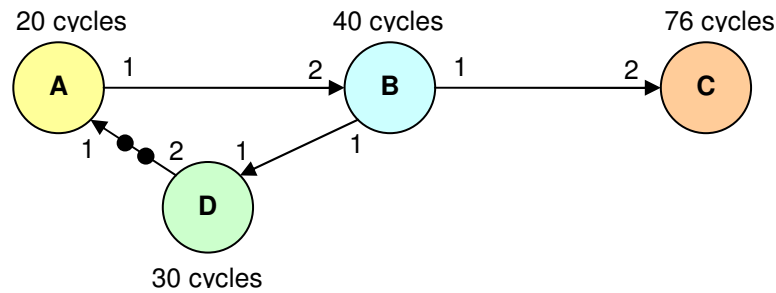


Figure 8 Synchronous Data Flow Graph

<sup>2</sup> In this thesis, the term “actor” and “task” are interchangeable, as we are modeling at this level. In data flow literature, the term actor is used, due to its more generic meaning.

A definition of SDF is given in [3] and [20], which is repeated here. An SDF graph can be presented as a tuple  $(V, E, d, P, O, I)$ , where

- $V$  is the set of actors.
- $E \subseteq V \times V$  is a set of directed edges.
- $d: E \rightarrow \mathbb{N}$  is a function describing the number of initial tokens for an edge.
- $P: V \rightarrow \mathbb{N}$  is a function describing the worst-case response time of actor  $v \in V$ . For this thesis, the worst-case response time is defined to be in cycles.
- $O: E \rightarrow \mathbb{N}$  is a function describing the number of tokens produced on edge  $(u, v) \in E$  by actor  $u$  for each execution.
- $I: E \rightarrow \mathbb{N}$  is a function describing the number of tokens consumed on edge  $(u, v) \in E$  by actor  $v$  for each execution.

An actor is allowed to fire if sufficient tokens are available on all incoming edges, hence the current buffer state  $b(t)$  being a vector over  $b_e(t)$  for each  $e \in E$  where  $t \in \mathbb{R}^+$  represents the time. At initialization, at  $t = 0$ , each edge contains the number of tokens specified by function  $d$ :

$$(\forall e: e \in E: b_e(0) = d(e)) \quad (1)$$

An actor  $v$  is allowed to fire (enabled) at time  $t$  under the condition that:

$$(\forall u: (u, v) \in E: b_{(u, v)}(t) \geq I((u, v))) \quad (2)$$

When an actor fires, it starts executing, it will consume tokens from the incoming edges. When execution finished, tokens are produced on the outgoing edges. This implies that multiple actors may fire simultaneously; showing parallelism is available in the computations. Therefore computations have to be free of side-effects, meaning communication between actors is only possible if explicitly modelled through edges.

A SDF graph is consistent, if a repetition vector  $f: V \rightarrow \mathbb{N}^+$  can be found, such that

$$(\forall (u, v) \in E: f(u) \cdot O((u, v)) = f(v) \cdot I((u, v))) \quad (3)$$

Intuitively this means that a firing sequence can be found, such that for each edge, the number of tokens produced is equal to the number of tokens consumed. For the graph in Figure 8, the repetition vector  $f(A) = 4; f(B) = 2; f(C) = 2; f(D) = 1$  can be found, which proves it is indeed consistent.

A special case is the homogeneous synchronous data flow (HSDF) graph, where the number of tokens produced and consumed on an edge is always one, hence  $(\forall e: e \in E: O(e) = I(e) = 1)$ . As a result,  $f(v) = 1$  for all  $v \in V$ . In a HSDF graph, every actor is fired exactly once before the schedule is repeated. Every consistent SDF graph can be transformed into an HSDF graph.

### 1.3.2 Cyclo-Static Data Flow

In [7], the CSDF is introduced as a more versatile model than SDF. CSDF retains the analysis properties of SDF and allows a static schedule to be created. The model is extended by introducing an instruction sequence for each actor  $v \in V$ , with function  $S : V \rightarrow \mathbb{N}^+$  representing the length of the sequence. Each element in the sequence is called a phase. During the  $n^{\text{th}}$  execution of actor, phase  $n \bmod S(v)$  is executed.

The following definitions are extended:

- $P : V \rightarrow \mathbb{N}^{S(v)}$  is a function describing the worst-case response time in cycles of actor  $v \in V$  for each phase.
- $O : E \rightarrow \mathbb{N}^{S(v)}$  is a function describing the number of tokens produced on edge  $(u, v) \in E$  by actor  $u$  for each execution.
- $I : E \rightarrow \mathbb{N}^{S(v)}$  is a function describing the number of tokens consumed on edge  $(u, v) \in E$  by actor  $v$  for each execution.

In Figure 9, a CSDF graph is represented.

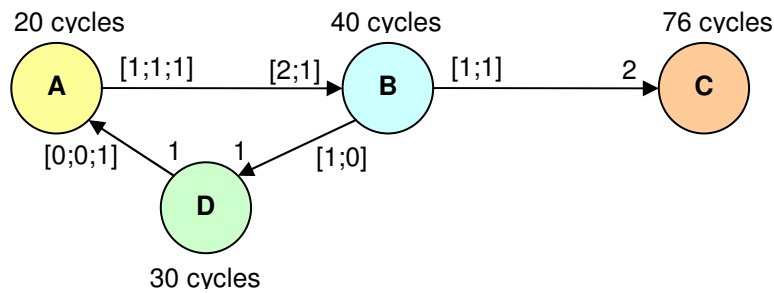


Figure 9 Cyclo-Static Data Flow Graph

### 1.3.3 Scheduling

The problem of assigning tasks onto a multiprocessor has been extensively studied in literature. With the introduction of SDF, Lee and Messerschmitt have also introduced a method for scheduling a task graph [15] [16]. When the actors in the data flow graph in Figure 8 are executed, as soon as their firing condition is validated and resources are available, the schedule as shown in Figure 10 can be derived. The schedule is repeated indefinitely and the execution party overlaps.

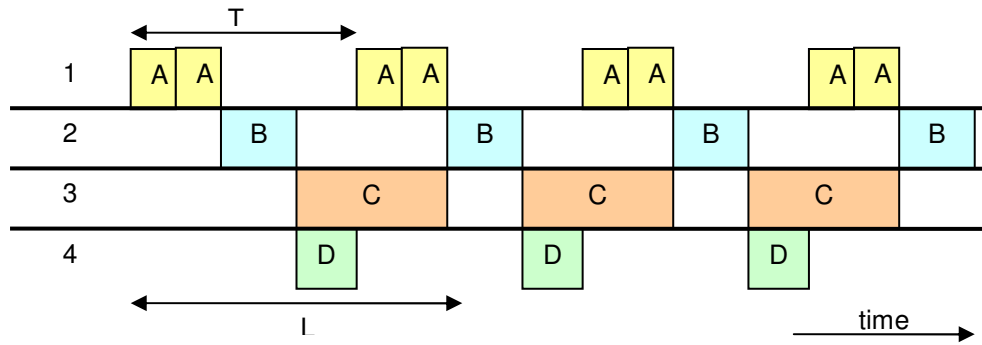


Figure 10 Schedule for a single period

The schedule is repeated every period of duration  $T$ , resulting in the throughput  $T^{-1}$  (iterations of the graph per time period). The period is limited by cycles in the graph and the availability of resources for execution. The latency for executing the entire task graph is denoted as  $L$ .

The scheduling algorithms for data flow graphs can be classified by which tasks are performed at compile-time (static) or at run-time (dynamic), as shown in Table 4 [24]. Which task can be performed at compile-time depends on the information which is available beforehand.

	Processor assignment	Actor ordering	Firing time
<b>Fully dynamic</b>	Dynamic	Dynamic	Dynamic
<b>Static assignment</b>	Static	Dynamic	Dynamic
<b>Self-timed</b>	Static	Static	Dynamic
<b>Fully static</b>	Static	Static	Static

Table 4 Scheduling classifications

For worst-case self-timed schedules it is proven in [20] that these will enter a periodic regime after a transitional period.

In [18] a scheduling strategy is presented to map multiple task graphs onto a multiprocessor system. As with the hardware model presented in this thesis, it considers a processor-local memory and shared memory. The scheduling strategy attempts to reduce communication overhead by clustering connected tasks and keeping them running on the same processing unit. An alternative method of scheduling aimed at creating a power-efficient schedule, using dynamic frequency and voltage scaling, are presented in [12], [17] and [26].

For dynamic schedules, care should be taken not to choose greedy locally optimal solutions [24]. A system controller is introduced, as explained in Section 4.5.2, which can perform global scheduling. The scheduling algorithm should be designed to provide globally optimally results.

### 1.3.4 Compact Representation

To provide a more compact representation for large CSDF graphs, two notational optimizations will be used in this thesis.

First, as or many edges  $e \in E$  it holds that  $I(e) = O(e)$ , the number will be placed on the edge only once, as shown in Figure 11.

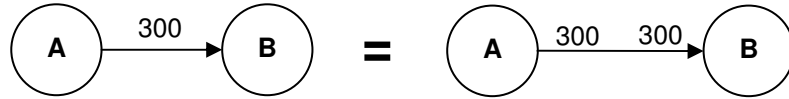


Figure 11 Compact Input-Output Notation

Secondly, the number of tokens produced by cyclo-static tasks is commonly the same for a number of phases. Therefore, if  $t$  tokens are communicated at each phase for the duration of  $n$  phases, it will be notated as  $[t^n]$ . If only a single number is given, this is the number of tokens communicated at every phase.

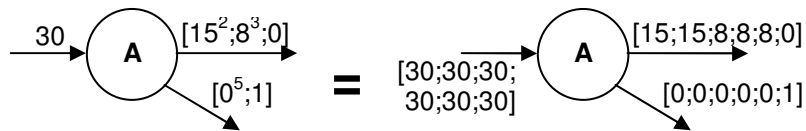


Figure 12 Compact Cyclo-Static Token Notation

## 2 Problem Statement

The cellular standards are rapidly evolving to provide higher data rates. The higher data rates have to be supported in the same bandwidths as previous standards, requiring more complex modulation techniques. As a result design of the baseband for a receiver faces several challenges:

- Increasing demand in computational power.  
While cellular standards allow faster communication, from 42 Mbps with HSPA+ to 300 Mbps with LTE and over 1 Gbps with LTE-Advanced. The mobile phone has to perform increasingly more complex calculations at a faster rate. This drives the need for faster hardware that can keep up.
- Increase in flexibility.  
The demand for flexibility is driven by the need to evolve with the standard and supporting many different standards. Today mobile phones need to be able to communicate using different wireless communication standards at the same time. The other demand is to have the mobile phone adapt to changes or enhancements made in the standards.
- Demand for higher energy efficiency.  
The applications become more complex at a faster rate than battery capacities improve. As a result, the same computations have to be done using less energy.

However, these requirements are conflicting with each other:

- Faster hardware is less flexible and more energy consuming.
- More flexible hardware is slower and more energy consuming.
- Energy efficient hardware is slower and less flexible.

### 2.1 Goals

The challenges in this thesis is to analyse the trade-offs between computational power and energy consumption in the modem. This requires exploring the various hardware architectures and the use of dynamic voltage and frequency scaling. As system becomes more complex, the optimal points for energy consumption become harder to predict. As the cost of manufacturing the actual hardware platform and time required for creating a real prototype are prohibitively expensive, the analysis should be performed beforehand. So, the hardware should be made the first time right.

For this project, the focus will be on the following elements:

- Exploring the behaviour of a system with different applications or in different modes of operation. For example, for LTE the workload scales heavily with the bandwidth used by the base station.

- Exploring the mapping of the algorithms on different hardware elements, such as the EVP or dedicated hardware. This includes the use of different scheduling policies.
- Explore various hardware architectures. The hardware architecture model has to be composed from using models of the EVP, DSPs, processors and dedicated hardware for processing units and different memory interconnect models for communication. The different hardware architectures should be simulated with dynamic frequency and voltage scaling.

At the end of the project, the following results should be delivered:

- Estimate power consumption using different hardware clock frequency and voltages. Determine communication trade-off against processing energy consumption.
- Investigate the use of dynamic processor assignment on a multiprocessor. Can energy consumption be reduced by using a dynamic scheduler?
- Analyze the dynamism of the application. The LTE cellular standard is designed to operate in various different bandwidths and MIMO modes. What are the effects on energy consumption and are improvements possible?

## 2.2 The Approach

To investigate various hardware architectures, software application and the power consumption a simulation model must be designed and implemented. The simulation model requires a methodology for constructing hardware architectures and describing software applications.

First, a methodology for exploring hardware architectures has to be chosen. A method for modelling has to be chosen that provides the right trade-off between accuracy, cost and speed of analysis. Three approaches seem possible, as show in Figure 13:

- Use mathematic models.  
The application and hardware can be modelled. As the behaviour of a system is analyzed, a model has to be found which models this aspect. The actual computations performed by the application are not relevant, especially since these ignore the implementation details, like computation time.  
Since power consumption under typical operating conditions is investigated, the model needs has to account for these uncertainties. In this case, stochastic models can be used. These will severely complicate the models.
- Using a simulation.  
With the use of a simulation, the behaviour of each individual component can be described independently. Only the interfaces between the different components have to be specified. Any behaviour caused by interactions between the components is determined when the simulation is running. This allows real implementations to be mimicked more closely and accurately.

- Perform measurement on the real implementation.  
Creating the hardware platform and software on a semiconductor is prohibitively expensive and time-consuming. Rather, the method is intended to explore the design before the real implementation is created.

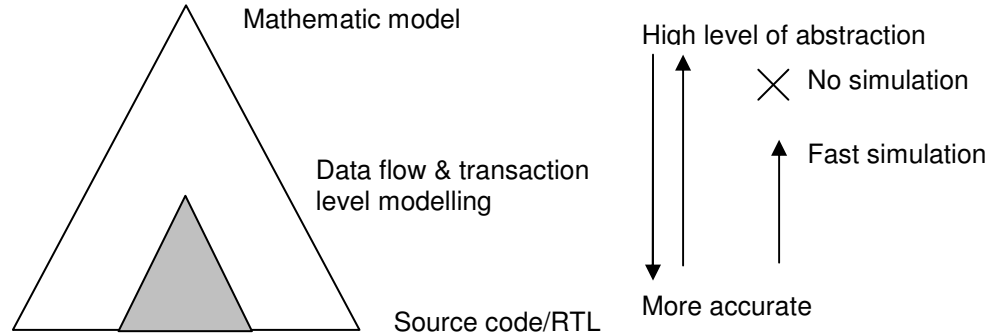


Figure 13 Levels of abstraction for application modeling

The simulation method is chosen for this project. For modelling and simulating an electronic system, CoWare, inc. provides a commercial-of-the-shelf tool: CoWare Platform Architect. This tool provides readily available cycle-accurate models for various industry-standard busses which can be used as the basis of the design. The tool builds on the SystemC modelling language, which is the industry-standard language for performing electronic system analysis.

To provide flexible system for performing architecture experimentation, software is developed for this project to automate creating of hardware platforms and mapping the application onto it. The software makes it simple for the user to generate many different hardware platforms and application models. The flow of the simulation is depicted in Figure 14. The results of the simulations are analyzed to improve the designed systems.

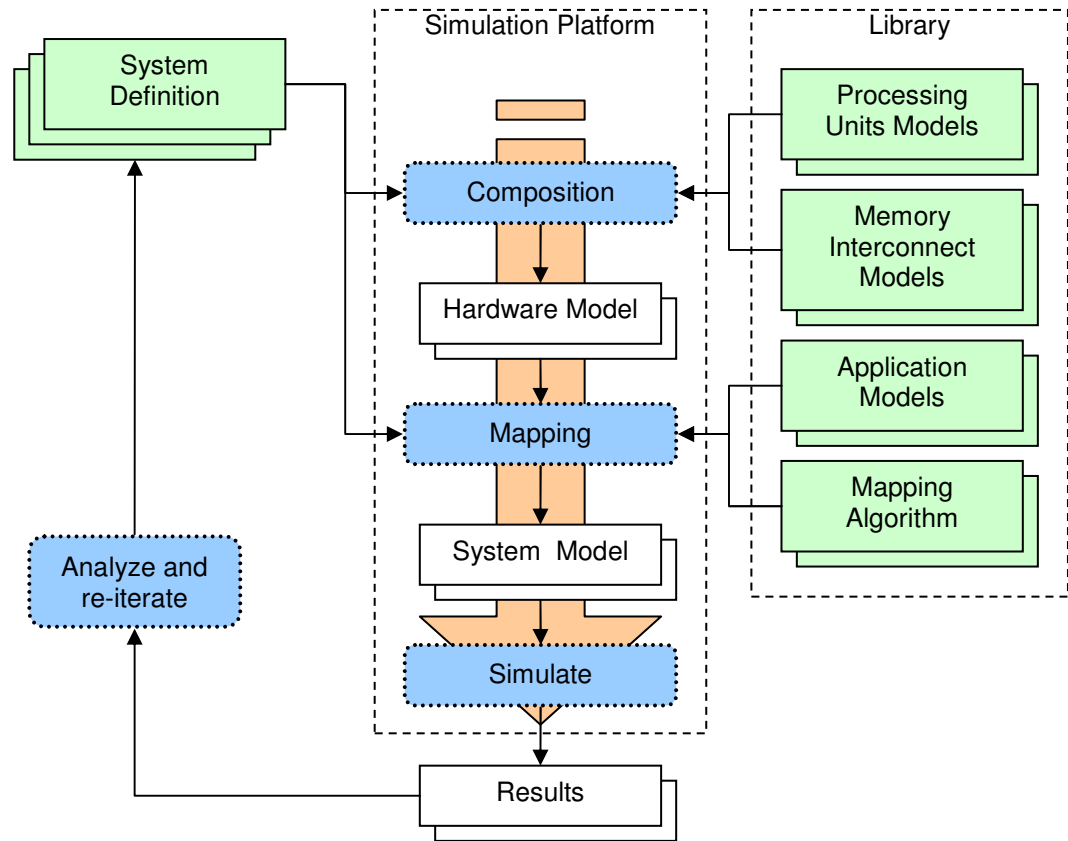


Figure 14 Simulation work flow

The main input for the simulation platform is a system definition, which specifies the assembly and configuration of:

- **Hardware model.**  
The hardware model is composed from processing units, the memories and interconnects. The processing units model the various processors, digital signal processors and dedicated computational hardware. The memory and interconnect enable communication between the computational elements. The hardware model is created in SystemC using CoWare Platform Creator. In Chapter 4 the hardware models are described.
- **Application model.**  
The application is modelled using Cyclo-Static data flow (CSDF) graphs. Using LIME, the graphs are compiled into a SystemC model, which is mapped onto the hardware model. In Chapter 3 the application modelling is described.
- **Mapping algorithm.**  
The algorithms for performing processor allocation. The algorithm dictates how and when scheduling decisions are being made. In Chapter 6 the mapping is described.

When a system model has been assembled, it is simulated and the results can be analyzed. The simulation provides the following insight in the system:

- Activity on the processing units.  
The processing units will be executing the application. The start and stop times of each task invocation is logged.
- Activity on the memory interconnects.  
The reads from and writes to the memory are logged and can be analyzed.
- Energy consumption of the system.  
The energy consumption depends on the current operating frequency, supply voltage and load of the processing units and the amount of data which is transferred. In Chapter 5 the measurement of energy is explained.

# 3 Application Modelling

This chapter gives an overview of the model of computation and the application modelled using it. The actual modelling is done with the tool Less Is More (LIME), as this enables the use of analysis tools already developed by ST Ericsson. To use the LIME model in the simulation environment, the tool is extended to generate code for CoWare platform architect, as explained in Section 3.1. The modelling and implementation of the tasks is explained in Section 3.2. In this study the cellular standard LTE is taken as a basis and the focus is on the inner receiver. The LTE task graph which has been created is explained in Section 3.3.

## 3.1 Task Graph Flow

The application is modelled in CSDF, as described in Section 1.3. The method is intuitive for decomposing the algorithm into separate elements. CSDF allows actor to contain many phases, which allow modelling of the resource symbols, which only appear in some ODFM symbols.

In Figure 15, the flow of the automated tools is depicted. The input is the CSDF task graph, including performance estimates and initial tokens. To transform the task graph into a SystemC model for Coware Platform Creator, it is first compiled by Less Is More (LIME). For this project, LIME is extended to generate the desired SystemC code. Then, the generated model is annotated with the execution times for the actors, the initial tokens on the edges and finally the maximum buffer sizes for the edges. The buffer sizes must be known when the edges are mapped into memory, since only a finite amount of memory is available.

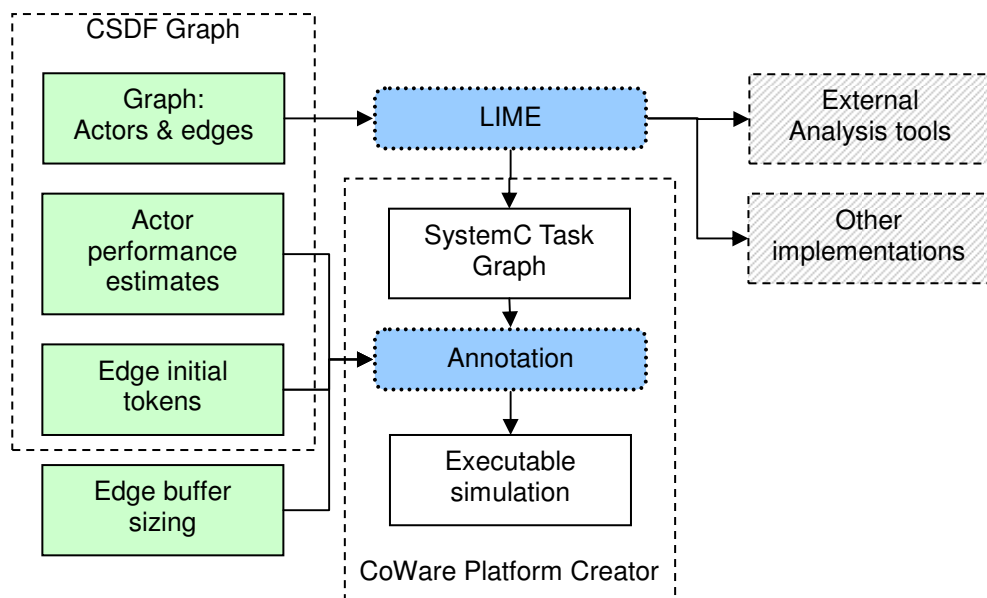


Figure 15 Application modeling tool flow

By using LIME, the task graphs can be analyzed by other tools or the graph can be reused from other implementations. The tools have been designed to allow external tools to be added to the workflow afterwards.

## 3.2 Task Model

All actors in the CSDF graph are transformed in SystemC tasks and channels, which can be simulated using CoWare Platform Creator. Starting with the CSDF actor, as shown in Figure 16, it has to be transformed into a sequential program. The implementation for the actor is described Section 3.2.1. The implementation of the edges, or FIFO buffers, is described in Section 3.2.4.

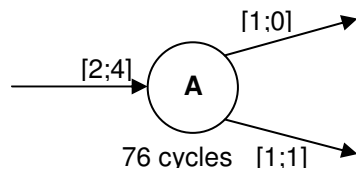


Figure 16 Single CSDF Actor

### 3.2.1 Conversion to Finite FIFO Buffers

Communication between two actors happens through a FIFO channel. In SDF and CSDF, the FIFO channels have an infinite capacity. However, in the real world they are mapped into a physical memory with only a limited capacity. As a result, all FIFO channel must have a bounded capacity.

The CSDF graph can be converted into a graph where buffer sizes are bounded [20], as shown in Figure 17. The buffer size constraint can be expressed by adding a back-edge for each existing edge. The number of initial tokens on the back-edge equals the buffer size.

According to the firing rules, an actor may only fire when all incoming edges contain sufficient tokens. In the top graph, A has no incoming edges and is therefore allowed to fire indefinitely. A back-edge is added to keep track of the number of available places in the FIFO buffer. As a result, actor A has an incoming edge and can only fire twice before actor B must fire and empty the FIFO channel.

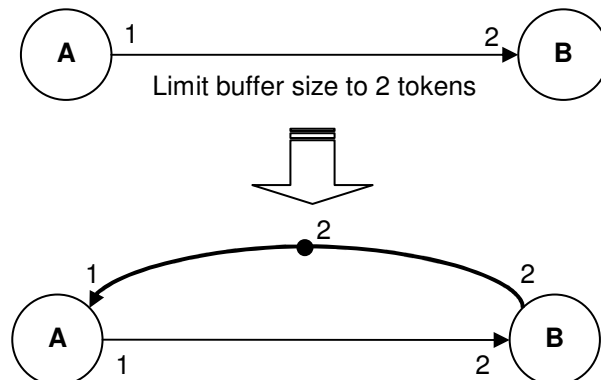


Figure 17 Transforming FIFO channel into bounded FIFO channel

For any CSDF graph the minimum required buffer sizes can be found, as described in [7]. However, there might be good reasons to use larger buffer sizes, as it might allow more freedom when executing the task graph or to compensate for a varying load. This decision cannot be made when the task graph is created.

### 3.2.2 SDF Task Implementation

Each CSDF actor is converted into a SystemC task. The task will continuously execute the following operations, as depicted in Figure 18:

- Check firing condition  
Before a CSDF task can execute, first the firing condition has to be checked. On all incoming edges sufficient tokens must be available. When translating to FIFO buffers, sufficient data has to be ready on incoming buffers and sufficient space has to be available on outgoing buffers. Only if the firing condition holds, the task is allowed to continue execution.
- Fetch data.  
The data has to be read from all incoming FIFO buffers.
- Execute algorithm.  
The task has to execute the algorithm and consumes cycles.
- Store data  
Data has to be written to all outgoing FIFO buffers.

When the firing condition has been validated, it is known that sufficient buffer space is available. This implies no memory constraints will prevent a task from successfully executing until all operations have been completed. Therefore, a good point to reschedule the task is after the operations.

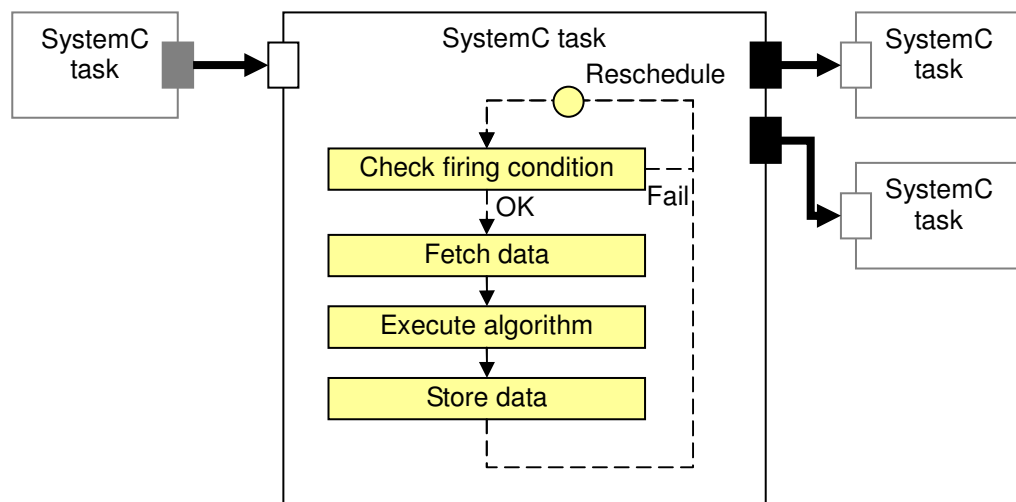


Figure 18 Task Implementation

### 3.2.3 CSDF Task Implementation

While the previous model would handle SDF graph, it does not yet handle CSDF actors with multiple phases. Therefore the execution path has to be extended to go through each phase sequentially, as shown in Figure 19. Therefore, if the firing condition for phase 1 has been validated and all operations for phase 1 has been executed, the task can perform the same operations for phase 2. When this has been done for all phases, it will return to phase 1 again.

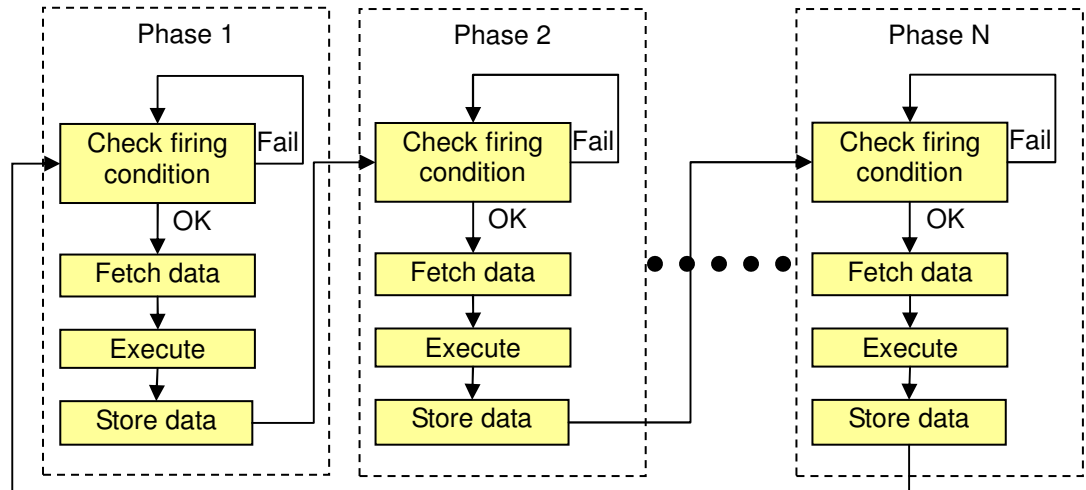


Figure 19 CSDF Task Graph Sequence

When a system controller (see Section 4.5.2) is introduced, a centralized entity will decide which task will execute on which processor. The system controller instructs the tasks which phase they must execute. Therefore, instead of checking the firing condition, the task has to wait for the command from system controller, as shown in Figure 20. After receiving the command, the requested phase will be executed and afterwards the task will wait for the next command.

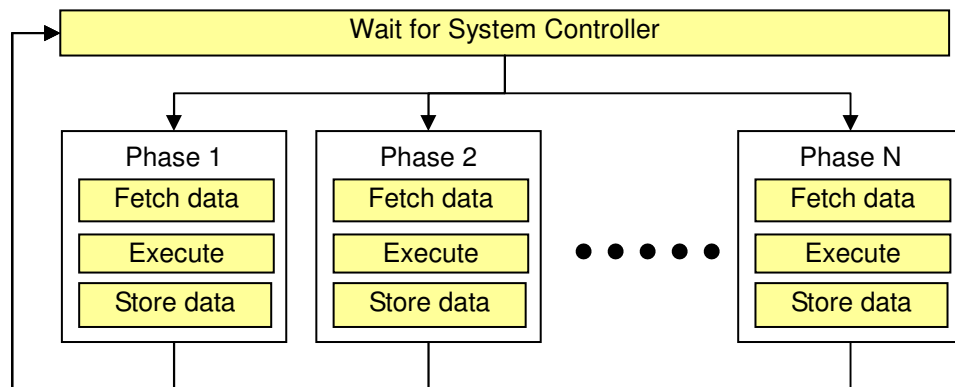


Figure 20 CSDF Task with System Controller supervision

However, once a task is started, it will execute all associated operations for that phase until completion. When a task instance is executing, only a single processing unit will be running it. This model is more limited than the pre-emptive execution which is common on personal computers.

### 3.2.4 FIFO Channel Implementation

The FIFO channels are the communication mechanism between different tasks. A FIFO channel differs from an edge in SDF and CSDF since it only has a finite capacity. As seen in Section 3.2.1, each edge can be transformed in a bounded size FIFO channel.

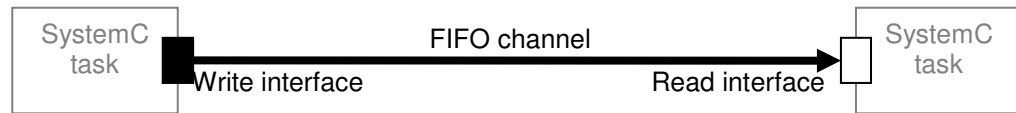


Figure 21 FIFO channel

The FIFO channel, as shown in Figure 21, provides two interfaces: a write interface for the producer and a read interface for the consumer. The implementation used in this simulation is based on Sea-of-DSP (SoD) FIFO interface. SoD is an existing framework for developing stream processing application on embedded hardware. It is developed by Philips and used by ST Ericsson.

The FIFO channel interface provides the following semantics, as shown in Figure 22:

1. The first step is checking the firing condition of the task. For CSDF graphs, it requires sufficient tokens are available on the incoming edges. Therefore, on the read FIFO, sufficient data should be ready. On the write FIFO, sufficient buffer space should be available. The available buffer space is indicated by the number of tokens on the back-edge of the write FIFO.
2. After the firing condition is validated, the task can read the data from the read FIFO.
3. After processing the data, the task writes the output to the write FIFO.
4. Finally, for the read FIFO it is signalled that the allocated memory can be reused. This is done by placing the read tokens on the back-edge of the read FIFO. The write FIFO is signalled to indicate to the next task that new data is available.

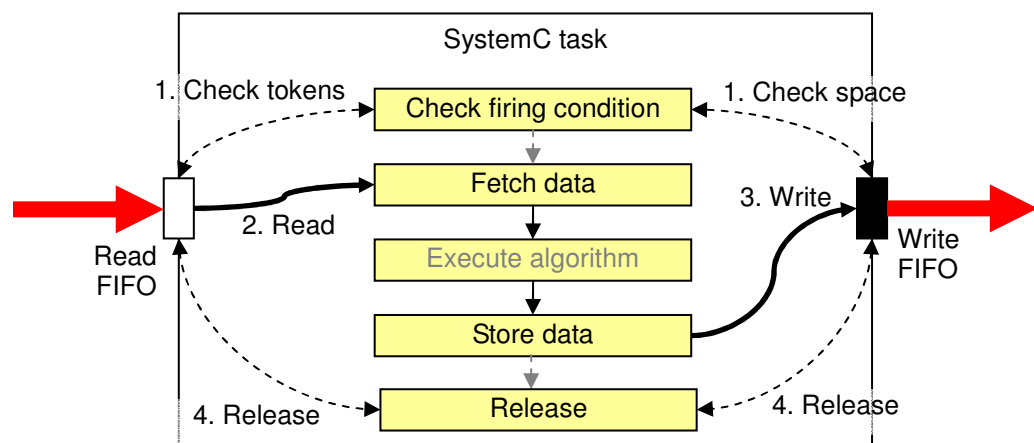


Figure 22 FIFO channel interaction

The advantage of using a generic interface is freedom in the implementation. For this project two implementations are provided:

- A single SystemC module implementing the entire FIFO channel.  
This allows independent task graph simulation. In addition, it is used to model the processor-local memory, since the FIFO channel remains internally in the VPU onto which the two communicating tasks will be mapped.
- A split read and write driver.  
The write driver accepts writes from the SystemC task and stores them in memory using the memory interface of the VPU. The read driver reads from the SystemC task and reads the data from memory using the memory interface of the VPU. This enables communication using a memory-mapped FIFO channel, enabling communication between tasks running on different processing units.

## 3.3 Long Term Evolution Task Graph

In this section the modelling of the Long Term Evolution (LTE) which has been created is described. For the application model, a cyclo-static data flow (CSDF) graph is used to specify it. The model and performance estimates are based on the work done by ST Ericsson (formerly NXP) for an LTE-capable software modem at the Dresden site [22]. The project aims to develop a product that supports LTE with 2 receive antennas and capable of receiving with speeds up to 150 Mbps.

### 3.3.1 Inner receiver overview

In Figure 23, the partitioning of the graph is shown. The inner receiver is being split into several partitions. The signal received from the RF and DFE is first processed by the antenna pre-processors. The antenna pre-processor performs the synchronization, channel estimation and demodulation. The communication between the antenna pre-processors is limited to performing noise normalisation. The next step is combining the pre-processed results and performing demapping. In MIMO transmissions, the independent data streams are separated from the antenna signal using the estimated channel function (see Section 1.1.2). The next step is to demodulate the signal to the bit-level and feed it to the error correcting algorithms in the outer receiver.

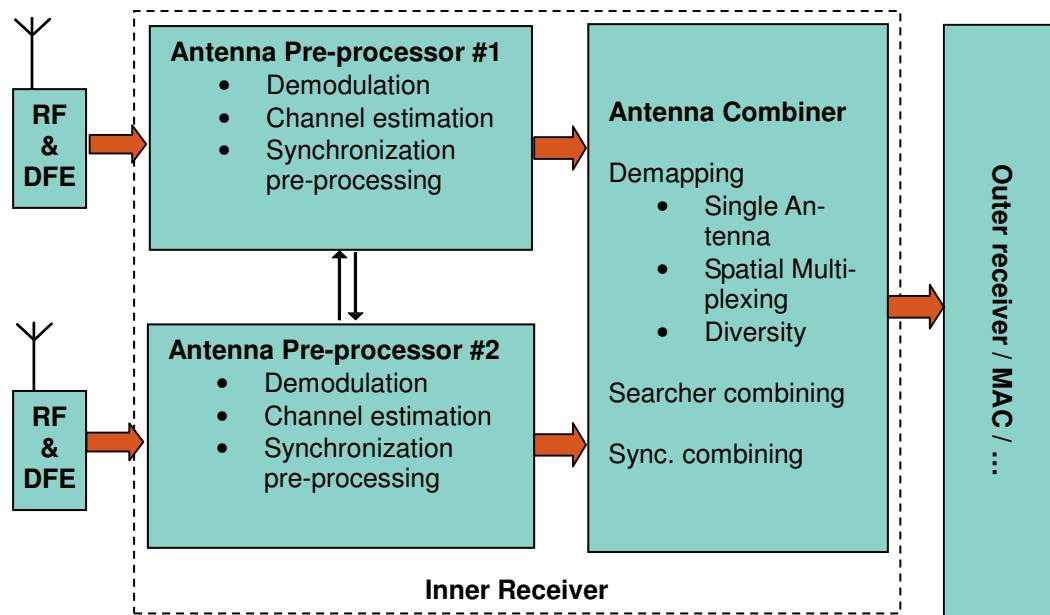


Figure 23 LTE Receiver Partitioning

The design of the inner receiver is based on 3 EVPs. Each partition is mapped onto an individual EVP, meaning 2 EVPs are used for the 2 antenna pre-processors and 1 EVP is used for the antenna combiner.

### 3.3.2 Dynamism of the application

The LTE standard specified different modes of operation [1]. The different bandwidth requirements mean different amounts of data to be processed, resulting in variations in the number of required cycles and different memory usage.

To express this, two parameters are introduced in the task graph, as shown in Table 5:

- K: the number of OFDM carriers.
- B: the number of FFT points.

The bandwidth is the bandwidth mode specified for LTE. Samples specifies the number of samples which are received by the RF/ADC and transmitted to the DFE. The FFT points are the number of samples which are demodulated. The number of samples is always proportional to the number of FFT points. After the FFT, the OFDM carriers are extracted, taking the number specified in the OFDM carriers value. The number of reference symbols a single antenna port transmits, are listed under reference symbols. These are always  $1/6^{\text{th}}$  of the number of OFDM carriers.

Bandwidth (MHz)	Samples	FFT points (B)	OFDM carriers (K)	Reference Symbols (K/6)
1.4	160	128	72	12
3.0	320	256	144	24
5	640	512	300	50
10	1280	1024	600	100
15	1920	1536	900	150
20	2560	2048	1200	200

Table 5 LTE Bandwidth and parameters  $K$ , the number of OFDM carriers, and  $B$ , the number of FFT points.

The LTE standard also supports different MIMO modes. The receiver will have 2 antennas, leaving the option of 2 or 4 antennas for the transmitter. This influences how many OFDM symbols contain reference symbols.

MIMO modes	Downlink (Mbps) @ 20 MHz
2x2	150
4x2	150

### 3.3.3 Antenna Pre-processor

The antenna pre-processor is responsible for conditioning the received signal and performing channel estimation. A simplified overview of the antenna pre-processor is shown in Figure 24. First, the received signal is demodulated using the FFT. All OFDM carriers are normalized and passed on to antenna combiner. The reference symbols are used to estimate the channels, which are used by the antenna combiner to extract the different MIMO streams.

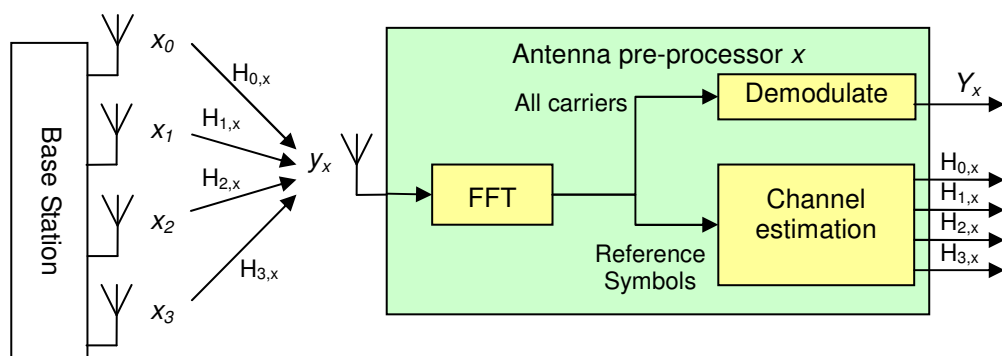


Figure 24 Antenna pre-processor

In Figure 25, the task graph for the antenna pre-processor stage is shown. The model includes the RF/ADC, automatic gain control (AGC) and digital front-end (DFE) for completeness. They should be mapped on dedicated hardware. As the RF/ADC task generates data for a single OFDM symbol, it must execute with a rate of 1 firing every 1/14 ms.

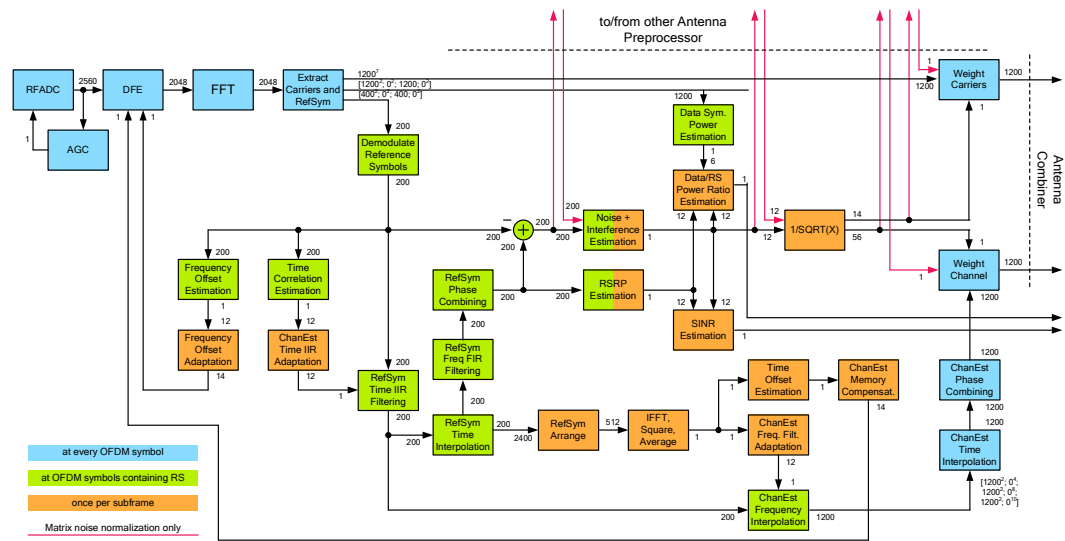


Figure 25 LTE Antenna Preprocessor Task Graph

The number of cycles consumed by the invocation of each task is shown in Table 6. The table is split into the RF/ADC and DFE part and the antenna pre-processor part. The following columns are given:

- The first column lists the name of the task.
- The second column shows the number of cycles which are consumed for each invocation of the task as a function of FFT size  $B$  and number of OFDM carriers  $K$ , as provided in Table 5.
- In the third column the number of cycles which are consumed per invocation are computed for the worst-case LTE bandwidth of 20 MHz.
- The fourth column gives the repetition vector  $f$  as described in formula (3). Since 14 OFDM symbols are transmitted per sub frame, we can determine the FFT will execute 14 times per sub frame. Therefore, we should solve  $f$  with  $f(FFT) = 14$ .
- In the fifth column the number of cycles consumed every sub frame is listed. It consists of multiplying the number of cycles per invocation times the repetition vector.

Task	Cycles / Invoke (function)	Cycles / Invoke	Invokes / Subframe	Cycles / Subframe
RF/ADC	n/a	100	14	1.400
AGC	$10/32 \cdot B$	640	14	8.960
DFE	n/a	100	14	1.400
<b>Total RF and DFE cycles / subframe</b>				<b>11.760</b>

Addition	$30 + 1/12 \cdot K$	130	12	1.560
ChanEstFreqFiltAdapt	300	300	1	300
ChanEstFreqInt	$30 + 30/8 \cdot K$	4530	12	54.360
ChanEstMemComp	$30 + 4/8 \cdot K$	630	1	630
ChanEstPhaseComb	$30 + 2/8 \cdot K$	330	56	18.480
ChanEstTimeIRAdapt	40	40	1	40
ChanEstTimeInt	$30 + 3/8 \cdot K$	480	56	26.880
DataPowerEst	$158 + 4/48 \cdot K$	258	6	1.548
DataRSPowerRatioEst	100	100	1	100
DemodRS	$30 + 4/8 \cdot K$	153	12	1.836
ExtractCarriers	$60 + 2/6 \cdot K$	460	14	6.440
FFT	$5/32 \cdot B \cdot \log(B)$	3520	14	49.280
FreqOffsetEst	40	40	12	480
FreqOffsetAdapt	100	100	1	100
NIEst	$30 + 5/48 \cdot K$	155	12	1.860
OneDivSQRTx	$40 + 10/48 \cdot K$	290	1	290
RefSymArrange	$30 + 1/8 \cdot B + 2/6 \cdot K$	686	1	686
RefSymFreqFIRFilt	$30 + 30/48 \cdot K$	780	12	9.360
RefSymIFFTSqAvg	$30 + 3/8 \cdot B + 5/32 \cdot B \cdot \log(B)$	4318	1	4.318
RefSymPhaseCombine	$30 + 3/48 \cdot K$	105	12	1.260
RefSymTimeIRFilt	$30 + 4/48 \cdot K$	130	12	1.560
RefSymTimeInterpolate	$30 + 5/48 \cdot K$	155	12	1.860
RSRPEst	$30 + 3/48 \cdot K$	105	12	1.260
SINREst	$30 + 5/96 \cdot K$	92	1	92
TimeCorEst	$30 + 4/48 \cdot K$	120	12	1.440
TimeOffsetEst	200	200	1	200
WeightCarriers	$60 + 73/193 \cdot K$	516	14	7.224
WeightChannel	$60 + 73/193 \cdot K$	516	56	28.896
<b>Total antenna pre-processor cycles / subframe</b>				<b>222.280</b>

Table 6 LTE Antenna Pre-processor Task Graph Cycle Times

### 3.3.4 Antenna combiner

The demodulated signals ( $y$ ) and the channel estimates ( $h$ ) are fed to the antenna combiner, as shown in Figure 26. The channel estimates are multiplied by the precoding matrix. The sphere decoder will separate the MIMO streams and perform the demapping. The result is sent to the outer receiver.

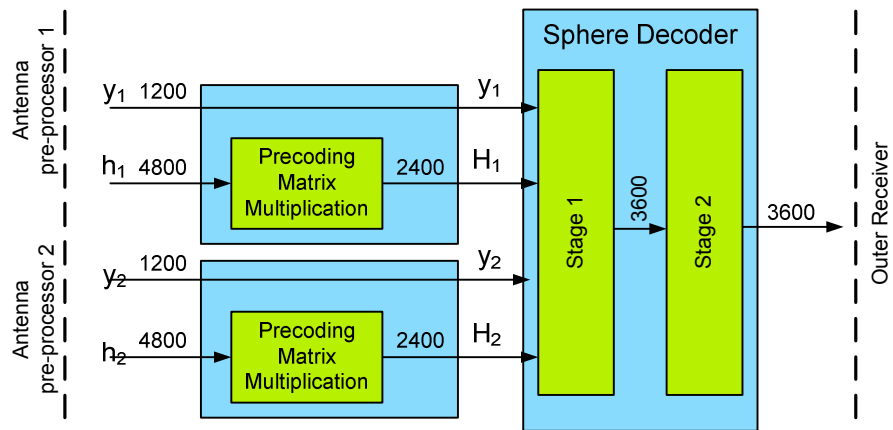


Figure 26 LTE Antenna Combiner Task Graph

In Table 7, the number of cycles consumed by each task is listed in the same way as done in Section 3.3.3.

Task	Cycles / Invoke (function)	Cycles / Invoke	Invokes / Subframe	Cycles / Subframe
Precoding Matrix Mult.	$30 + 9/8 \cdot K$	1380	$2 \cdot 14$	38.640
Sphere; stage 1	$30 + 45/8 \cdot K$	6780	14	94.920
Shpere; stage 2	$30 + 25/8 \cdot K$	3780	14	52.920
<b>Total Antenna combiner cycles / subframe</b>				<b>186.480</b>

Table 7 LTE Antenna Combiner Task Graph Cycle Times

# 4 Hardware Modelling

This chapter describes the model for the hardware which has been designed. Section 4.1 describes how the hardware is composed. Section 4.2 describes the tools which are developed for generating the hardware. The hardware architecture consists of processors, digital signal processors and dedicated hardware for computations, which are described in section 4.3. The communication between these processing units and the memories is described in section 4.4. Finally, each processor provides facilities for the tasks it will be executing, which is explained in Section 4.5.

## 4.1 Hardware Composition

The hardware model is responsible for modelling the delays caused by the execution of the application. An example of a hardware platform is given in Figure 27, showing several different processing units, which will be executing the application. The processing units can represent processors, such as the embedded vector processor (EVP), but also dedicated hardware, such as the RF/ADC or digital front end (DFE).

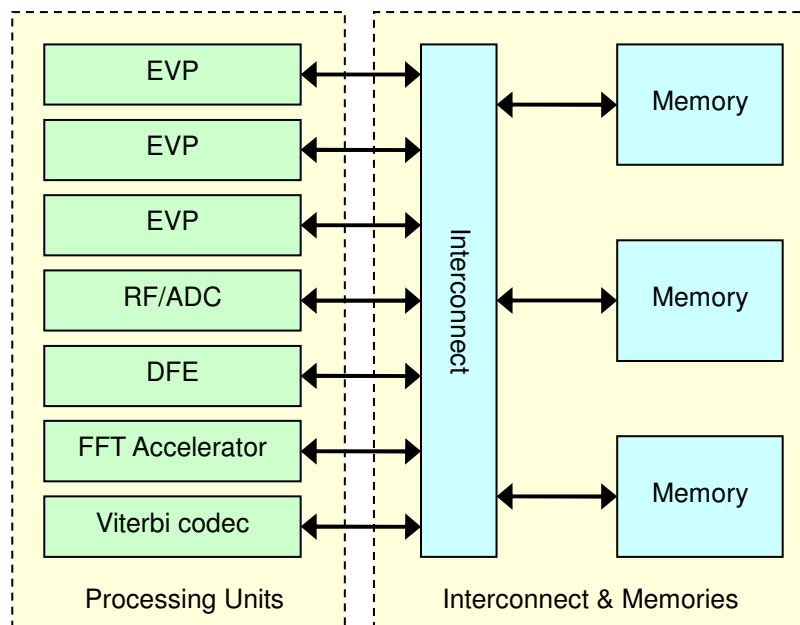


Figure 27 Hardware model overview

The processing units are responsible for allowing a task to execute. Each processing unit can execute a single task at a time. With multiple processing units available, many tasks can be executed in parallel. The interconnect and memory allow communication between the tasks.

The most important property of the hardware is the modelling of the delay time. When a task is executing on a processing unit, it will fetch data, execute its algorithm and store the data (see Section 3.2). As shown in Figure 28, the execution time of the task depends

on speed of the processing unit. The fetch and store times depends on the speed of the interconnect and memory.

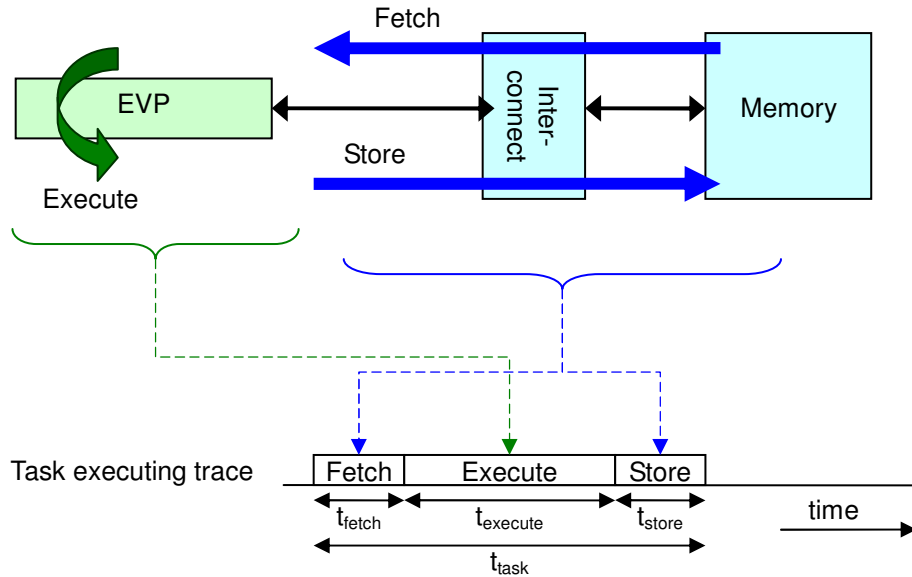


Figure 28 Hardware timing model

## 4.2 Hardware Flow

The hardware is modelled using CoWare Platform Creator. The Platform Creator uses SystemC for the models, which is an industry-standard language for modelling language. It provides a simulation kernel, allowing many different hardware architectures to be tested. CoWare provides models for a virtual processor and existing industry-standard busses. These can be used as the foundation for a system design where many of the implementation details are abstracted away. The CoWare Platform Creator provides a graphical user interface for creating and modifying the model. In addition, the entire process can be automated using TCL scripts.

For this project, as shown in Figure 29, TCL scripts to generate parameterized models for the processing unit and the interconnect and memories are developed. The simulation tool executes these scripts and generates the hardware in CoWare Platform Creator. The hardware model can be simulated. This allows the quick creating of many different systems, enabling efficient hardware exploration.

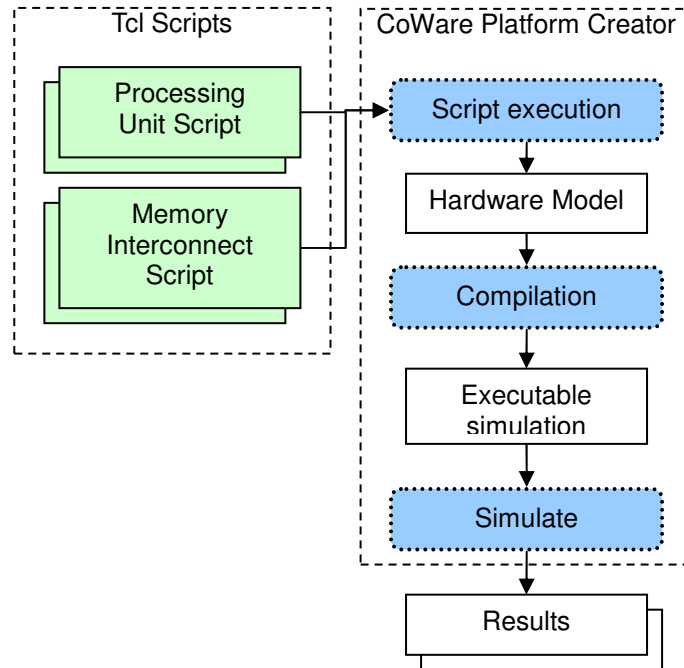


Figure 29 CoWare Hardware creation flow

## 4.3 Processing Units

A typical hardware platform for a mobile phone consists of many computational resources. These resources vary in capacity and capability. At one end there are dedicated accelerators, which are tuned for a single algorithm. There are digital signal processors, such as the EVP, which allow a wider variety of algorithms to be executed. And finally the more generic microprocessors, which are more suited at executing control code. All the hardware shares two characteristics:

- Performing computations takes time and ties up its resources.
- Processing happens on data, which has to be retrieved and transmitted.

Since only the timing behaviour of the application is of interest, the details of the hardware can be abstracted. The processing unit model only has to account for:

- Model time delay caused by executing an algorithm.
- Generating memory reads and writes.  
The delay will be decided by the interconnect and memories.

In Figure 30 the model for the virtual processing unit is given, as provided by CoWare. It contains an interface to communicate with the memory. A task manager controls selection between the different tasks. The execution time of the task depends on the clock input, which decides the speed of the processing unit.

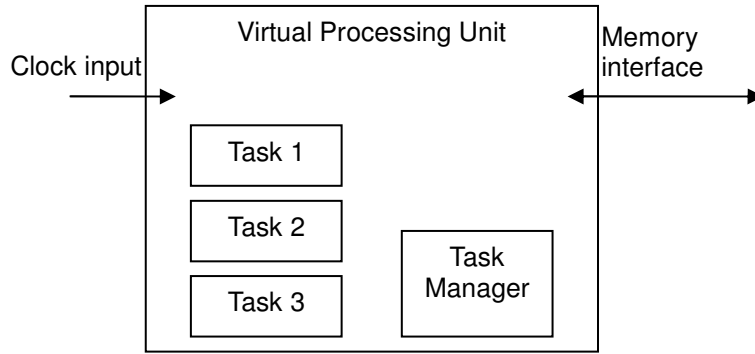


Figure 30 Virtual Processing Unit Model

Using the clock input and the task execution time, the real execution time can be derived. The task (or actor)  $v \in V$  has worst-case execution time  $P(v)$ , which is defined to be in cycles. The cycle time can be defined as  $t_{cycle}(n)$  for the  $n^{\text{th}}$  cycle. If task execution starts at cycle  $s$ , the execution time will be

$$t_{execute}(v, s) = \sum_{i=s}^{s+P(v)-1} t_{cycle}(i) \quad (4)$$

For a fixed-rate clock input the formula can be simplified, with frequency  $f_c$  and cycle

time  $t_{cycle}(n) = T_c = \frac{1}{f_c}$  for all  $n$ , then the executing time will be

$$t_{execute}(v) = P(v) \cdot T_c \quad (5)$$

Besides executing a task, several facilities have to be provided to support multiple tasks and scheduling. These are explained in Section 4.5.

## 4.4 Interconnect and Memory

This section describes the model build for the interconnect and memory design. The different tasks which are running the processors will have to communicate with each other. Following the SDF model, all communication goes through First In First Out (FIFO) buffers. For the simulation platform it is decided to map all FIFO buffers into memory, as shown in Figure 31. Communication between the processing units and memories happens through an interconnect. Therefore, the interconnect dictates the communication times.

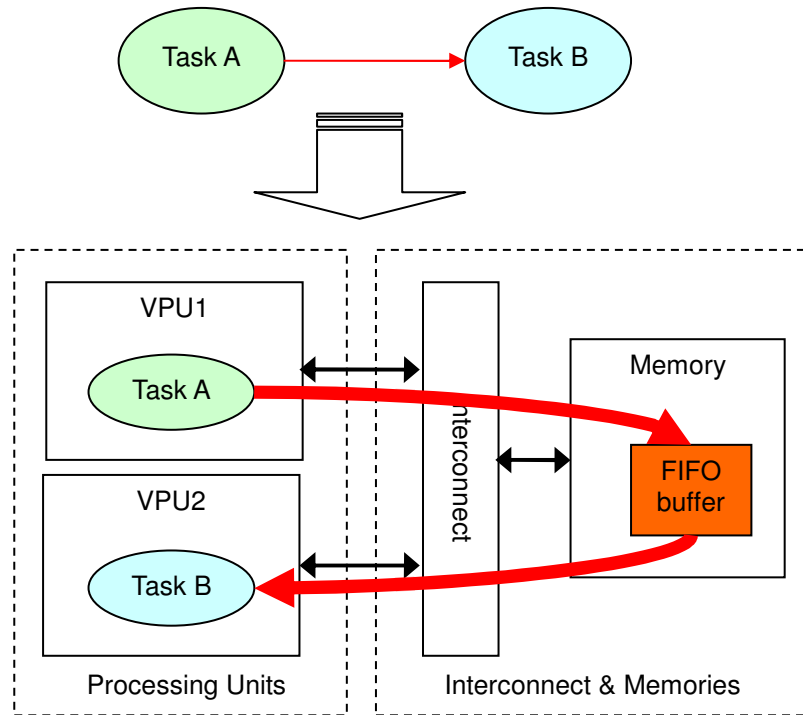


Figure 31 Communication using shared memory

This section will describe several interconnect designs. The processor-local memories are discussed in Section 4.4.1. The shared bus, which is widely used, is explained in 4.4.2. The Uniform Access Memory, an interconnect and memory design which allows parallel access from all processing units is discussed in Section 4.4.3. Finally the use of caches is discussed in Section 4.4.4.

#### 4.4.1 Local memory

In the hardware model every processing unit will have its own processor-local memory. In the model the local memories are not explicitly modelled. Instead, when two tasks on the same processor have to communicate directly, the FIFO channel implementation is used (see Section 3.2.4). The use of the FIFO channel implementation means no memory accesses to the shared memory are required. As a result, no time is spent on transferring the information to another task.

#### 4.4.2 Shared bus

The most common interconnect between a processor and memory is the shared bus model. In Figure 32, a shared bus with four VPUs, which can request memory reads and writes, and a single memory, which provides responses. A single transport resource is available which is shared between all processors and memories. Only a single processor can send or receive data at a time.

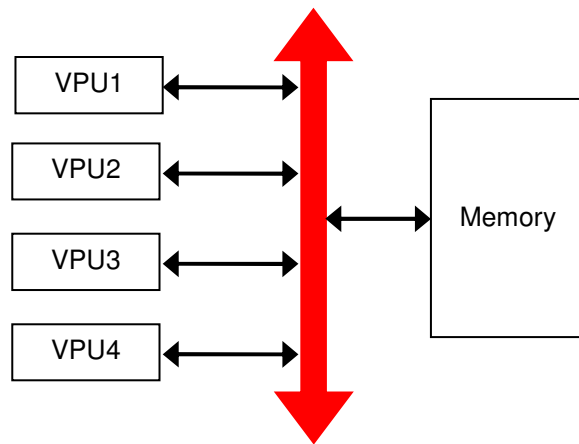


Figure 32 Shared bus model

Before a processor can send or receive data over the shared resource, access must first be granted by an arbiter. Typically, the arbiter will grant access for a specific duration, called the burst size. This allows a sequence of words to be transmitted in quick succession. The arbiter is free to choose a scheduling policy. The use of a round robin policy provides a fair distribution between all connected processors. In addition, for real-time systems it is possible to establish an upper bound, making analysis possible. We will assume an arbiter which implements round robin scheduling policy in this thesis.

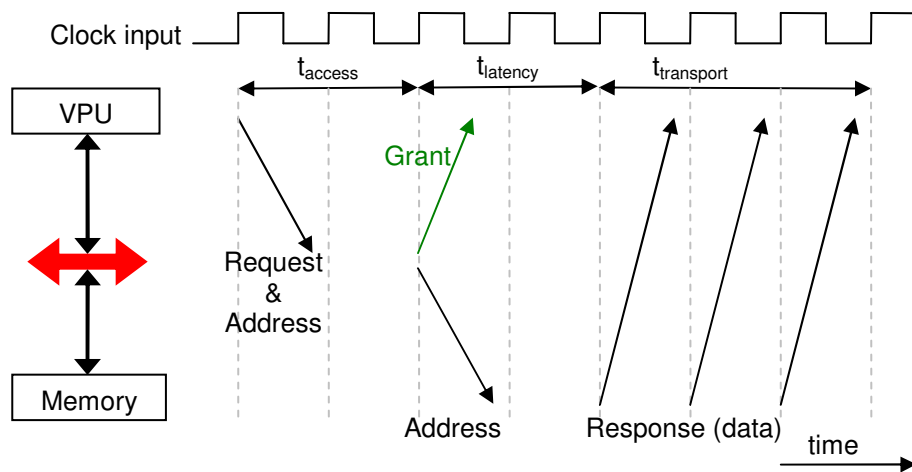


Figure 33 Shared bus timing model for reads

The message flow requires for accessing memory over a shared bus are depicted in Figure 33. First, the processing unit needs to gain access to the bus, since it might currently be in use by another processor. The second step is to send the address to be read, or the address and data to be written. In the last step the memory sends back the read data, or acknowledges the write. The interconnect is able to transfer a single word per cycle. The delay for an access can be described by the following formula, where  $s$  indicates the size of the data in words,  $m$  the number of masters on the bus and  $b$  indicates the burst size:

$$t_{shared}(s, m, b) = t_{access}(m, b) + t_{latency} + t_{transport}(s, m, b) \tag{6}$$

The access time depends on the state of the bus. If the bus is inactive, the master may gain immediate access. Hence the best case access time is:

$$bct_{access}(m, b) = 0 \quad (7)$$

In case of a round robin model, where every cycle another processor gains access to the bus, the worst case access time is:

$$wct_{access}(m, b) = \frac{(m-1) \cdot b}{f_{bus}} \quad (8)$$

The latency is the time it takes to transmit the address and retrieve the data. The latency largely depends on the memory. The latency is denoted by  $t_{latency}$ . The latency is typically fixed, since it does not depend on the state of the bus.

Next, data has to be transferred over the bus. As the bus can transfer a single word every cycle, the best case time to transfer the data is:

$$bct_{transport}(s, m, b) = \frac{s}{f_{bus}} \quad (9)$$

In the worst-case scenario, all other masters are also transferring data, resulting in the bus being used to transfer data to other masters and slaves. In between the transfer, a wait time is inserted to allow others masters to transfer data. In this case, the worst case transfer time becomes:

$$wct_{transfer}(s, m, b) = \frac{s}{f_{bus}} + \frac{(\lceil s/b \rceil - 1) \cdot (m-1)}{f_{bus}} \quad (10)$$

At this point the best-case and worst-case time can be determined from (6), (7), (8), (9) and (10):

$$bct_{shared}(s, masters) = \frac{s}{f_{bus}} + t_{latency} \quad (11)$$

$$wct_{shared}(s, m, b) = \frac{s + (\lceil s/b \rceil + b - 1) \cdot (m-1)}{f_{bus}} + t_{latency} \quad (12)$$

The difference between worst case time and best case varies largely as more masters are added to the system. Therefore the exact times required for communication are hard to determine.

For shared busses, CoWare provides readily available models for most industry-standard busses. These include the OCP bus, and ARM AMBA eXtensible Interface (AXI). These busses are fully parameterized and can be configured with any desired latency, burst size and number of masters.

### 4.4.3 Uniform Access Memory

In the Uniform Access model, parallel access to a single memory is provided to all processors. In contrast to the shared bus, the processors don't interfere with each other when accessing the memory. It allows high-speed access and is fully predictable.

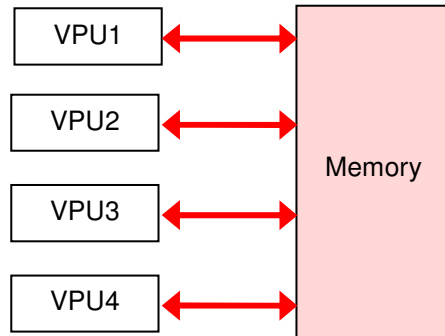


Figure 34 Uniform Access Memory model

Every processor has a dedicated bus, which connects to the memory. These busses can be viewed as a shared bus with a single master. Therefore, the best case and worst case times for uniform access memory become equal (for every burst size):

$$t_{uam}(s) = wct_{shared}(s, 1, b) = bct_{shared}(s, 1, b) = \frac{s}{f_{bus}} + t_{latency} \quad (13)$$

However, the memory architecture has to support the parallel access by all VPUs. Three possible design options are:

- Increase memory frequency.

If the memory frequency is equal to the VPU frequency times the number of connected processors:  $f_{memory} = f_{vpu} \cdot \#processors$ , it becomes possible to serve all processors. The advantage is that it allows fully random access and latencies do not increase. However, this will result in very high operating frequencies. If four processors running at 312 MHz are connected to memory, it requires the memory to operate at 1.2 GHz. As the processors will be designed to run at the maximum frequency, this goal seems unreachable.
- Multi-port memory.

A memory can be designed with multiple ports. The operating frequency does not change and all processors have fully random access. However, a dual-port SRAM has half the density and nearly twice the power consumption [21]. Therefore the solution does not seem attractive for a low-power system.
- Wide memory.

By using a wider memory, where more bits are transferred every cycle, it becomes possible to reach the desired data rates. If the memory width is the bus width times the number of processors, all processors can be served. However, the reads and writes have to be sequential to benefit from the wide memory access. In

addition, the reads or writes for a single processing unit can be handled in a cycle, which will increase the latency.

The use of extremely high frequencies is unachievable for cellular chips, leaving the multi-port and wide memory solutions. A wide single-port memory provides twice the capacity of a dual-port memory with the same area and power budget, as shown in Table 8. For memory with more ports, the required area and power is expected to increase quadratic with the number of ports. In [9] the area requirements for various multiport register files are presented, as shown in Table 9. Therefore the wide memory is the most interesting option for low power designs.

Technology node	Mbit/mm <sup>2</sup>	
	Single-port	Dual-port
45nm [21]	1.6	0.83
32nm	3.1	1.6
22nm	6.6	3.5

*Table 8 Memory area for different technology nodes*

The wide and multi-port memories require a large number of pins, making them unsuitable for off-chip implementation. More pins require larger chips and more wires on the PCB, while space in a mobile phone is highly constraint. Therefore an on-chip memory is required to provide a uniform access memory.

Memory ports	Relative Area	Capacity/Area	Capacity/Area/Port
5	1.00	1.00	1.00
6	1.38	0.72	0.87
7	1.73	0.58	0.81
8	2.21	0.45	0.72

*Table 9 Memory scaling for single-cycle multi-port register file*

The application running on the processing units will perform communication using FIFO buffers which are mapped in memory. This results in sequential reads and writes from and to the memory. The wide memory suits this goal, as random access is not required.

The uniform access memory can be implemented, as shown in Figure 35, by connecting the processing unit to an intermediate buffer. The buffer provides a bridge to the wider bus connected to the memory. The buffer can keep the processing unit bus occupied while it is waiting to access the wider memory-side bus.

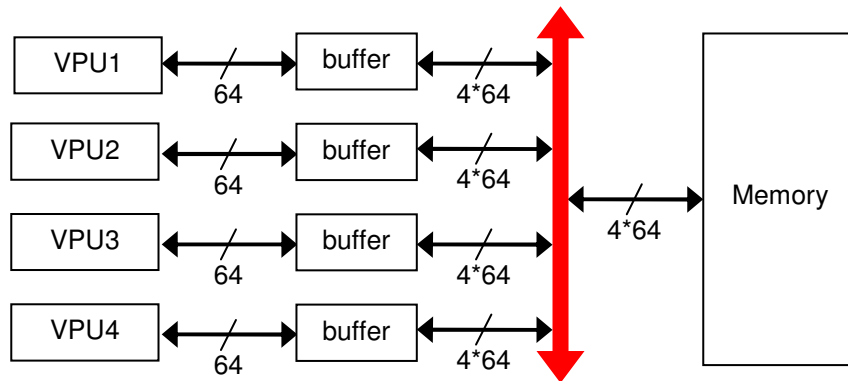


Figure 35 Uniform Access Memory Implementation

If the wide shared bus serves the buffers in a strict round-robin fashion, the minimum worst case latency becomes:

$$wct_{latency} \geq \frac{\#masters}{f_{memory}} \tag{14}$$

The uniform access memory can be modelled in CoWare Platform Architect using three available bus models, as shown in Figure 36:

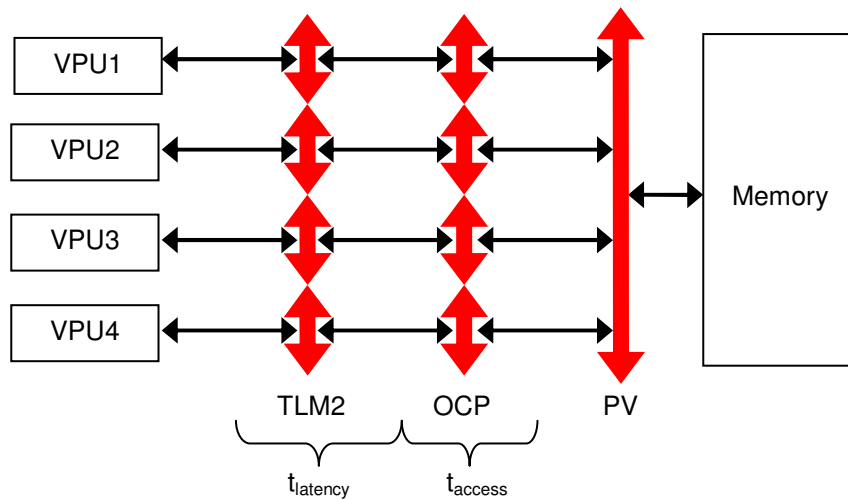


Figure 36 CoWare implementation for Uniform Access Memory

- The VPU issues memory transactions. The memory transaction contains the memory address and the amount of data to be read or written.
- The TLM2 bus adds the latency time to the entire transaction. This bus is used to model the latency time.
- The OCP bus splits the memory transaction into bus width units and simulates the one cycle per bus unit delay. This bus models the access time.
- The PV bus is untimed and therefore allows all accesses in parallel, simulating independent parallel access from the processors PV to the memory.

The implementation model allows the user to select any of the three design options.

#### 4.4.4 Memory Caches

The design of shared memory multiprocessors has also been discussed in [10]. The system uses a single memory which is visible to all processors. However, in order to improve the performance of the memory system, caches are introduced. These caches temporarily store the data, so it is readily available when the processor will read the same data again.

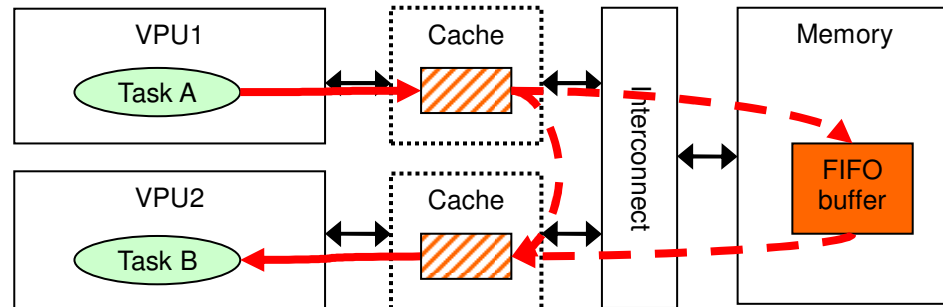


Figure 37 Cache Hierarchy

Introducing a cache provides the following advantages:

- Repeated reads can be retrieved from cache, resulting in faster responses and fewer memory accesses.
- Writes are stored in the cache, reducing memory accesses when multiple writes to the same location are performed.
- Some caches allow memory to be transferred between caches, reducing the number of memory accesses.

The caches provide the following disadvantages:

- Additional latency must be introduced to check the contents of the cache. For application with many cache misses, a cache may even degrade performance.
- For real-time application the analysis becomes more complex. As worst-case situations have to be assumed for real-time analysis, introducing caches may result in the inability to guarantee the real-time requirements.
- Guaranteeing cache coherency demands special protocols, which might increase overhead.

In this system, only FIFO buffers are placed in the memory. Data is written sequentially into the FIFO buffers, after which the data will be read only once. So there is no advantage in using a cache.

Even though memory access might be reduced, as some caches support transfers directly between the caches, all disadvantages are still valid. Therefore, the introduction of a cache does not make sense for the modelled application and will not be explored.

## 4.5 Software facilities

The virtual processing units provide several facilities to the tasks they will be executing. CoWare provides a task manager, which is explained in Section 4.5.1. To enable global scheduler, a system controller is introduced and is explained in Section 4.5.2.

### 4.5.1 Task manager

The Virtual Processing Unit provided by CoWare provides an integrated task manager. The task manager is responsible maintaining the task state, selecting a task to run and removing tasks from the waiting queue when their waiting condition is satisfied.

The different task states supported by the task manager<sup>3</sup> are shown in Figure 38. As each processing unit can only execute a single task at the same time, only one task can be in the running state. When the task has been executed, it gives control back to the task manager and becomes ready to run. Alternatively, it may indicate it has to wait for a certain event, such as an external signal and/or timeout, in which case it is placed in the waiting queue. A task is removed from the waiting queue and placed in the ready to run when the waiting condition has been satisfied.

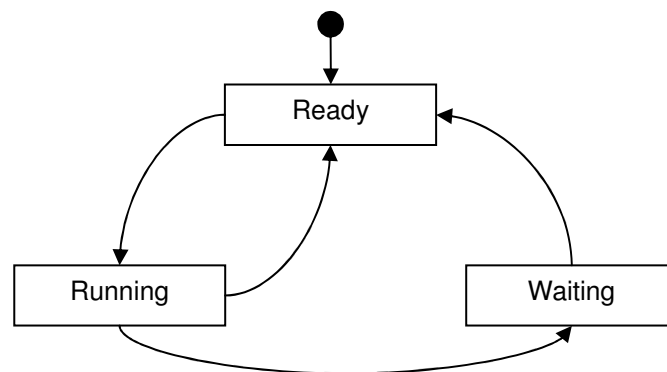


Figure 38 Task states

When no task is in the running state, the scheduler must pick a new task from the ready to run queue. As a result, the scheduler determines the actor ordering. By default, the tasks are executed in the order in which they enter the ready queue. This results in a dynamic actor ordering.

### 4.5.2 System Controller

The limitations of the CoWare provided task manager is that it only allows greedy task scheduling on the local processor. Once the task scheduler is invoked select the next task to run, it must pick a task in the ready queue. In embedded systems the widely used approach is to assign each processing unit a dedicated set of tasks. In this case, the local task scheduler is sufficient. However, as tasks are allowed to be executed on different processing units, a global scheduler is desired. A global scheduler has an overview of all resources and can make better informed scheduling decisions.

<sup>3</sup> The CoWare task manager support more states, however they are not used in this work and therefore left out.

Two methods of implementing a global scheduler are considered:

1. Use of a system controller.
2. Use of a distributed scheduling algorithm.

The system controller provides the simplest implementation. The system controller makes all scheduling decisions for all processing units, resulting in a single entity to keep track of all the information

The distributed algorithm, while allowing the use of the existing local task manager, but it requires a more complicated algorithm. In addition, a distributed algorithm requires the data needed to make scheduling decisions to be available to every instance. As a result, extensive communication to provide all instances with the desired data is required.

	System Controller	Distributed scheduler
Algorithm	Simple	Complex (distributed)
Communication	Start & finished messages	All data for decisions
Task manager	Extended	Heavily extended

Table 10 System Controller vs Distributed Scheduler

On a real hardware platform, generally a simple low-power processor is used for scheduling and control. Scheduling algorithms mainly consist of making decisions, for which a microcontroller is best suited. A digital signal processor, such as the EVP, should rather be kept busy with tasks it does best: number crunching. In addition, there is no need to add additional logic to hardware accelerators for scheduling.

Therefore, a dedicated system controller is chosen. A single VPU is elected as system controller and performs scheduling and starting tasks remotely, as shown in Figure 39.

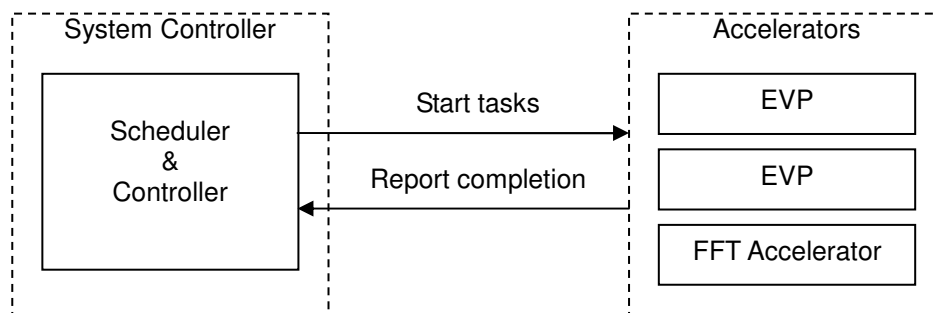


Figure 39 System Controller System Interaction

In this way, the system control interface works like a remote procedure call. The system controller invokes the algorithm on a remote processor and awaits completion. The start command contains the task and phase identifier and the buffer locations. This gives the system controller full control over the operation of the accelerators.

The system controller is an extension to the virtual processing unit by adding a remote procedure call (RPC) subsystem, as shown in Figure 40. The RPC subsystems are linked though a communication bus. The first VPU is selected as the system controller and is

responsible for executing the global scheduler. The second VPU also contains 4 tasks which are under control of the global scheduler.

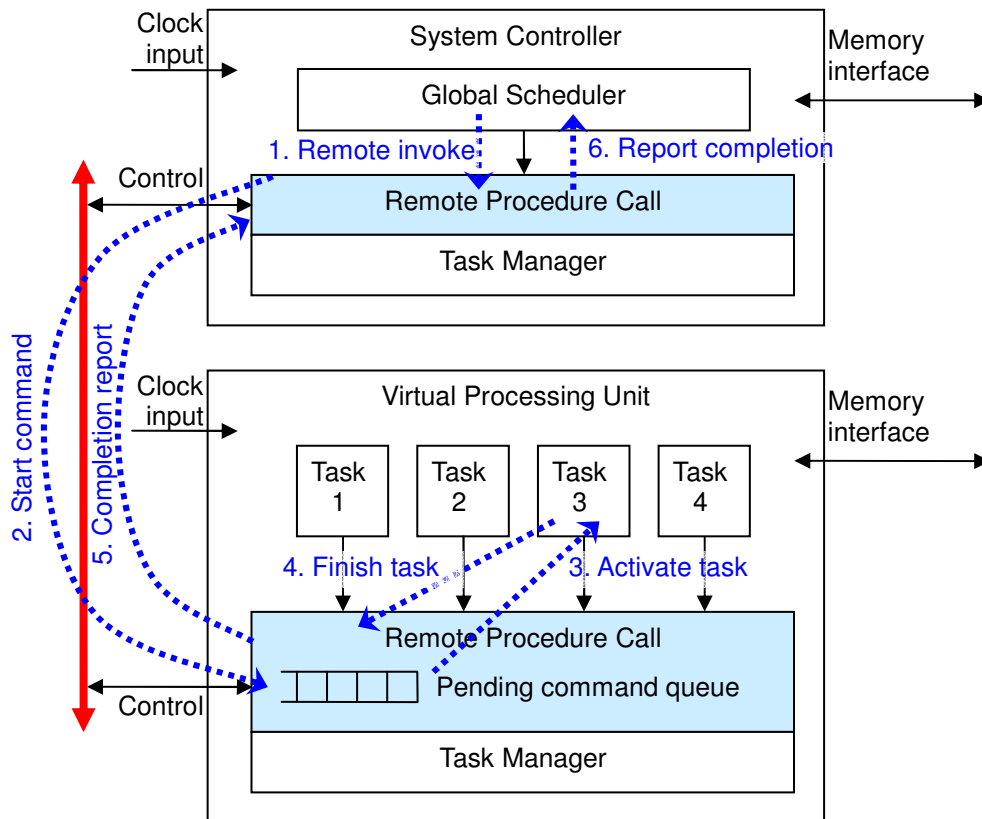


Figure 40 Virtual Processing Unit Extensions for System Controller support

The following actions are executed:

1. The global scheduler directs the RPC subsystem to start task 3 on the second VPU.
2. The RPC subsystem transmits the start command over the communication bus to the second VPU. The start command contains all the relevant information, including the task identifier, the phase and the memory locations of the FIFO buffers. The RPC subsystem queues the start command and signals the task manager.
3. The task manager can select the appropriate time to run the task and active it. As soon as task 3 is started, the start command is dequeued and the relevant parameters are sent by the RPC subsystem to the task. At this point, the task starts executing.
4. As soon as the task has finished, it informs the RPC subsystem executing has finished and the global scheduler can be informed.
5. A completion report is transmitted to the system controller, where the start command originated.

6. The RPC subsystem informs the global scheduler that the task has finished execution.

The design of the system controller enables the placement of tasks on the system controller VPU. In addition, multiple global schedulers can be supported.

# 5 Energy reporting

The goal of the thesis is analyzing the power consumption when the designed hardware is running an application. The hardware model, as shown in Figure 41, consists of processing units and interconnect driven by a global clock source. In order to support dynamic frequency and voltage scaling, clock dividers are introduced, which allow the operating frequency to be reduced. There is a fixed relation between supply voltage and operating frequency.

During the simulation, the current divider value, activity on the processing units and interconnect are logged. By combing the operating mode and the activity of the processing unit, their energy consumption can be computed. By analyzing the memory activity taking place, the energy consumption for communication is computed.

In Section 5.1 the model used for energy computations is introduced. In Section 5.2, the model is extended for dynamic voltage and frequency scaling. In Section 5.3, the power measurement for the processing units and in Section 5.4, the power measurement for the interconnect and memory is explained.

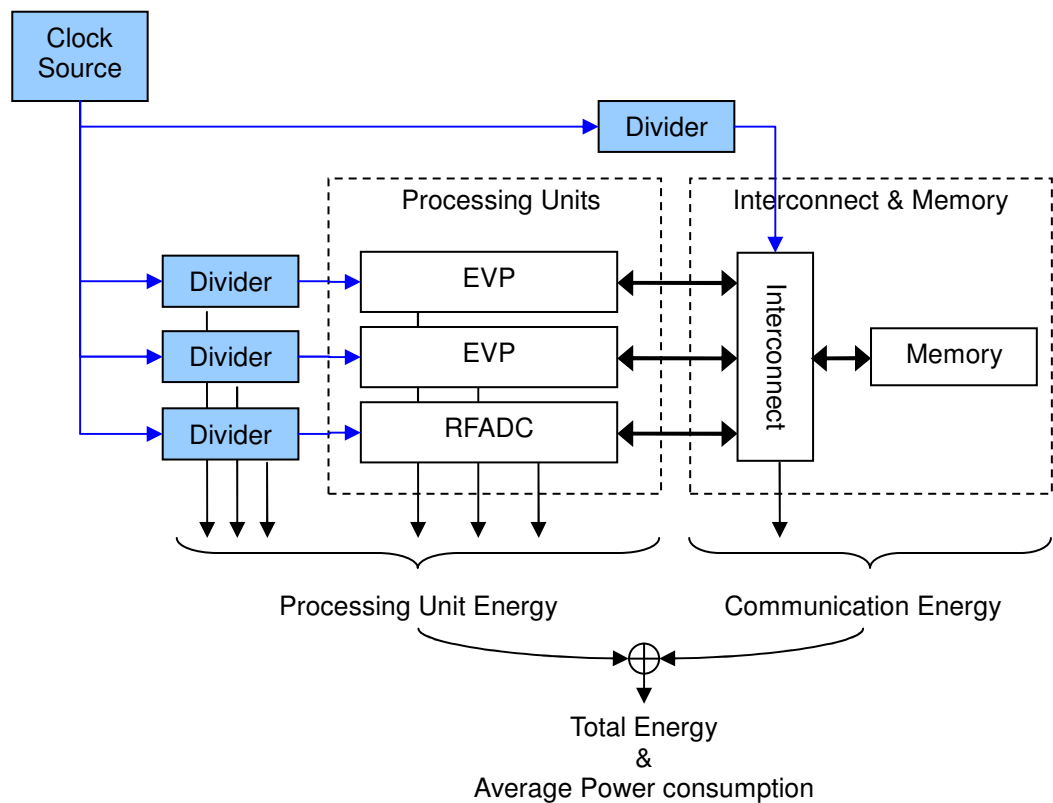


Figure 41 Power Measurement and Dynamic Frequency and Voltage Scaling Model

## 5.1 Energy model

To measure the energy consumption of the processor and memories an energy model is required. In the energy model for CMOS, as described in [13], the two main sources of power dissipation are static current, which results from resistive paths between power supply and ground, and dynamic power, which results from switching capacitive loads between different voltage levels.

$$P = P_s + P_d \quad (15)$$

Since a low power semiconductor manufacturing process is used for the production of cellular chips, the static power consumption is very low and therefore neglected in this thesis. The equation for the dynamic power consumed by a CMOS transistor is:

$$P_d = aCV^2f \quad (16)$$

where  $a$  the activity factor,  $C$  the total capacitance of the output node,  $V$  the supply voltage. The energy per cycle can be derived from this formula, where cycle time  $T = \frac{1}{f}$ :

$$E_d = \int_0^T P_d dt = aCV^2 \quad (17)$$

For a CMOS circuit, the energy is the sum of all gates:

$$E_d = \sum_i a_i C_i V^2 = C_{eff} V^2 \quad (18)$$

This shows the energy dissipation of the system is the sum of the individual parts. Therefore power can be analyzed for each individual processor and memory interconnects. Afterwards these values can be added.

When a processing unit is executing an algorithm, the energy consumed depends on the amount of cycles  $d$  the algorithm takes:

$$E_d(d) = \sum_{i=1}^d C_{eff} V^2 = dC_{eff} V^2 \quad (19)$$

From this formula it can be derived that the energy requirements for executing an algorithm solely depends on the number of cycles, the effective capacity of the processor and the supply voltage. The operating frequency of the circuit does not directly influence the costs of an algorithm.

## 5.2 Dynamic Frequency and Voltage Scaling

In order to improve energy efficiency, many modern processors employ dynamic frequency and voltage scaling. By reducing the voltage and/or frequency, the dissipated power decreases. However, equation (19) indicates the frequency does not influence the energy required for executing the same algorithm. Only lowering the voltage will improve the

energy efficiency of the same processing unit. However, there is a relation between the voltage and operating frequency. Using the simple  $\alpha$  power model the maximum delay

$$\frac{1}{f} \geq \tau = K \frac{V}{(V - V_{th})^\alpha} \quad (20)$$

where  $\tau$  is the delay of a gate,  $K$  is a proportionality constant specific to the technology and  $V_{th}$  the threshold voltage, with  $V_{th} < V$ . The  $\alpha$  is a constant to account for the fact that transistors may be velocity saturated, with  $1 < \alpha < 2$ . By lowering the voltage  $V$ , while the threshold voltage remains constant, the delay  $\tau$  increases. As a result, the frequency  $f$  must decrease.

In real products where low power consumption is a major concern, the supply voltage must be set to the lowest required for operating at the desired frequency. Following the computations, the supply voltage and operating frequency can be described as a monotonic function of each other (within their operating boundaries). The maximum frequency is limited by the voltage, using  $\frac{1}{f} = \tau$ . Then the energy consumption is defined

by the operating frequency alone. Therefore, it is sufficient to provide a facility to change the operating frequency. The minimum operation voltage will be derived using equation (20).

In Figure 41, the entire model is operated by a base frequency  $f$ , generated by the clock source. Rather than using multiple clock sources, the base clock signal passes through an clock divider, which produces an clock edge after  $n$  input edges. The output of the clock divider will switch at frequency  $f_n = \frac{f}{n}$ , where  $n \in \mathbb{N}^+$  is the divider value programmed into the clock divider.

With the introduction of the clock divider, the energy required for operation on the hardware driven by the hardware running on this clock can be determined. From the output frequency of the clock divider  $f_n$ , the minimum required voltage can be determined using equation (20), resulting in a monotonic decreasing function  $V_n = V(f_n)$ , where  $V_1 = V$  and  $n \in \mathbb{N}^+$ . The formula for dynamic power becomes:

$$E_{d,n} = C_{eff} V_n^2 = C_{eff} V_1^2 \frac{V_n^2}{V_1^2} = E_{d,1} \left( \frac{V_n}{V} \right)^2 = E_{d,1} R_n \quad (21)$$

For every clock divider value, the energy can be expressed as a ratio relative to the energy consumption at the base frequency.

## 5.3 Processing Unit Power

The processing unit is responsible for executing the tasks. The task executing consists of reading data, processing the data and writing it back to memory. When no task is executing the processor remains idle until a task is scheduled. As seen in equation (18),

the dynamic energy consumed by the processing unit per operation depends on the effectively switched capacity.

A decision has to be made on how much detail will be required while measuring the dynamic power consumption of the processing units. The execution of different parts of the algorithms might result in different energy consumption. It will even be dependent on the data which is being processed, which is not modelled at all and highly unpredictable. The simulation is to provide analysis of the system before the implementation; therefore the exact values cannot be given. Instead, it will be sufficient to use the average power consumption of the processing units. The variations in power consumption will be approximated closely enough by using averages instead.

A first-order model is to distinguish between just two processor states:

- Active: consuming power.
- Idle: no power if the clock is stopped.

The energy consumption of the processor can be determined by integrating the current power consumption of the processor over all periods where the processor is active. The current power consumption can be derived from equation (21) by:

$$P_{d,n} = E_{d,1} R_n f_n = \frac{E_{d,1} f_1 R_n}{n} = \frac{P_{d,1} R_n}{n} \quad (22)$$

The power consumption for between  $t_1$  and  $t_2$ , where  $div(t)$  is the current clock divider value is:

$$E = \int_{t_1}^{t_2} P_{d,div(t)} dt \quad (23)$$

## 5.4 Interconnect and Memory Power

The memory follows a similar model as the processing units. Whenever data is transferred, this activity will consume power. As the amount of data transferred is logged, this data can be used to compute the memory energy consumption.

Unlike the processing units, the supply voltage for memory can typically not be lowered. Therefore, the energy per operation will not vary when the frequency changes. For memory it is sufficient to analyze the bus throughput. The energy consumption is given in the energy per unit of data. By multiplying the throughput with the energy per unit, the power consumption is derived.

# 6 Mapping

This chapter will describe how the tasks in the application are mapped onto the platform, which involves assigning a task to a processing unit for execution as shown in Figure 42. The main mapping problem is deciding to which processors which tasks should be assigned and in which memory to place each FIFO buffer. These decisions can be made statically, allowing them to be computed and analyzed beforehand, or dynamically, where the decisions are made during run-time. In this thesis, the dynamic assignment of tasks is investigated.

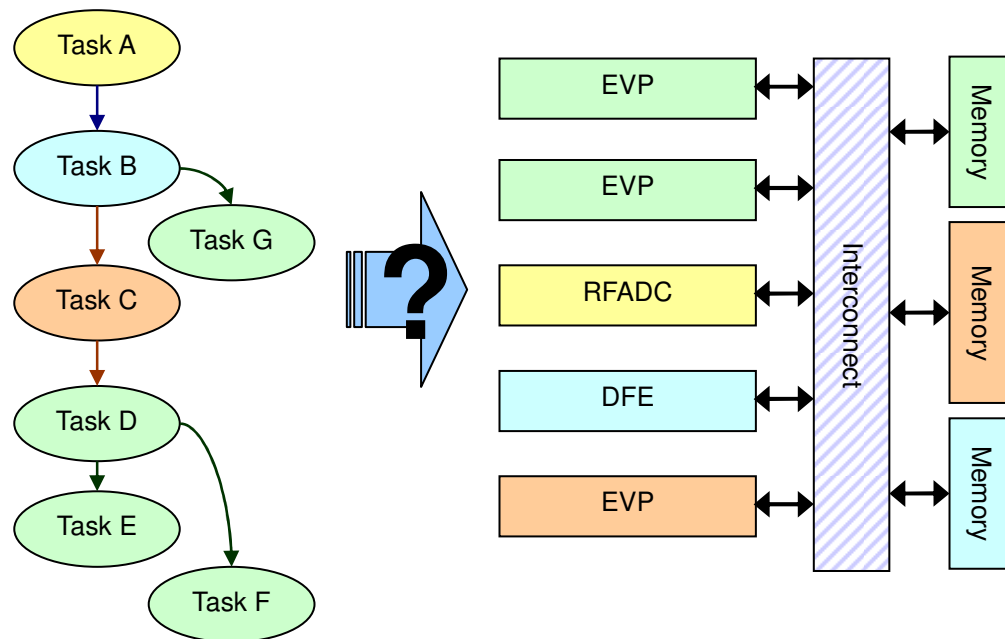


Figure 42 Mapping process

In Section 6.1 the operations performed by mapping algorithm is explained. An extension to the scheduling classifications is given in Section 6.2. In Section 6.3 the mapping process for performing static processor assignment is explained. In Section 6.4 the mapping process for dynamic processor assignment is explained. In Section 6.5, an improvement for dynamic processor assignment, supporting dynamic memory assignment is explained.

## 6.1 Mapping Overview

After a platform has been assembled and the task graph has been loaded, the mapping algorithm has to handle the following concerns:

- Assigning each task to a processing unit for execution. The system definition specifies the processor assignment. This is described in Section 6.1.1.

- Mapping each FIFO channel into a memory. The memory assignment has to be derived from the performed processor assignment. This is described in Section 6.1.2.

On the simulation platform, the firing times are always determined at run-time, meaning they are dynamic. As a result, a fully static schedule cannot be created. The actor ordering is handled by the task manager, as explained in Section 4.5.1.

### 6.1.1 Processor assignment

The first problem is mapping the tasks onto the processors. The system definition specifies the processor assignment. The mapping algorithm will instruct CoWare Platform Architect to assign the task to a VPU and place it under control of its task manager. Additionally, a connection to the remote procedure call interface of the task manager is created.

With dynamic processor assignment, a task is allowed to run on any available VPU. Since CoWare VPUs do not support moving tasks between different VPUs, an alternative approach is required. To solve this problem, a instance of the task is assigned to each VPU, as shown in Figure 43.

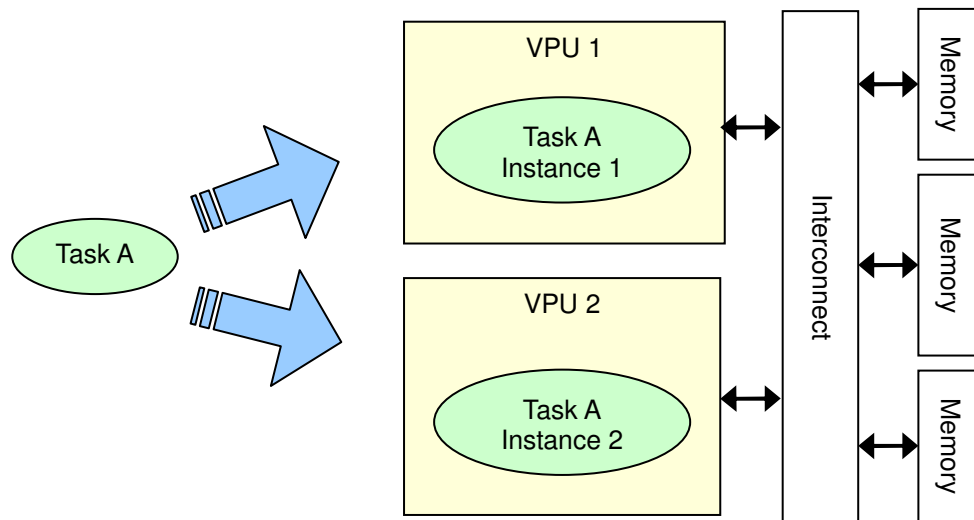


Figure 43 Dynamic processor assignment in CoWare

### 6.1.2 Memory assignment

The second mapping problem is memory assignment. As systems become more complex, more memories will be introduced. Processor-local memories are introduced to allow fast and low latency memory access by a single processor. To enable communication between different processors, an on-chip SRAM or external DDR SDRAM is used. These memories provide significantly more capacity, but have a higher latency and lower throughput. This memory hierarchy with advantages is depicted in Figure 44.

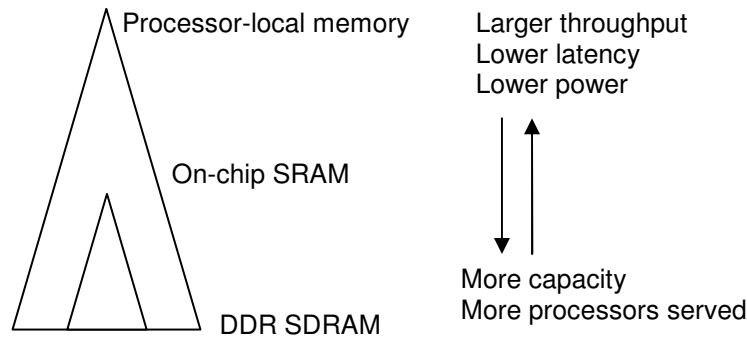


Figure 44 Memory hierarchy

The FIFO channels between the tasks have to be assigned to a memory. The assignment can be done at compile-time (static) by the mapping algorithm or at run-time (dynamic) by the global scheduler.



Figure 45 Simple task graph

Consider a simple task graph consisting of tasks A and B and a single FIFO channel from task A to task B, as depicted in Figure 45. The task graph is mapped onto hardware with a shared memory and processor-local memory. Two mapping options exist:

- Map tasks onto a single processing unit, shown in Figure 46. When both tasks will execute on the same processor. In this case the processor-local memory is the best choice. In CoWare this is modelled by using a FIFO channel internally in the processor.
- Map tasks to different processing units, shown in Figure 47. If the tasks reside on different processing units, the FIFO buffer has to be mapped onto a memory reachable by both processing units, therefore the shared memory is used. To enable communication through the memory interfaces in CoWare Platform Architecture, drivers must be added to each task.

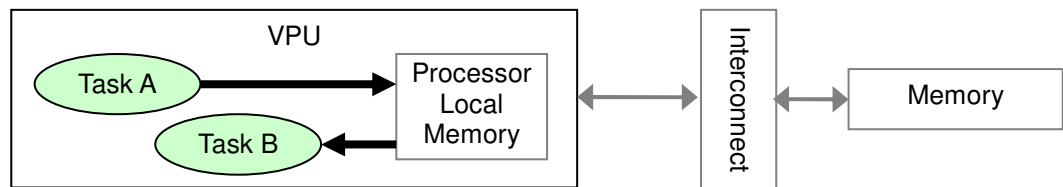


Figure 46 Task communication through Processor-Local Memory

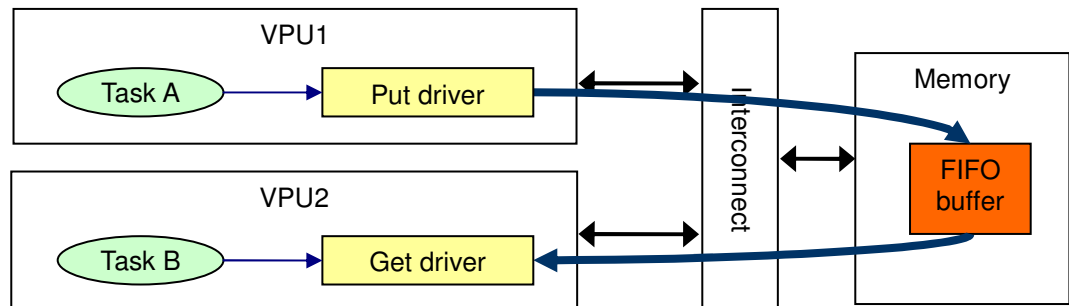


Figure 47 Task communication through Shared Memory

## 6.2 Classification for scheduling algorithms

With memory assignment being a prominent part of the mapping and scheduling, it becomes important to consider this aspect. The classification for scheduling algorithms, which are presented in Table 4 (on Page 21), has to be extended with memory assignment, as shown in Table 11.

	Memory assignment	Processor assignment	Actor ordering	Firing time
<b>Fully dynamic</b>	Dynamic	Dynamic	Dynamic	Dynamic
<b>Dynamic task assignment</b>	Static	Dynamic	Dynamic	Dynamic
<b>Static assignment</b>	Static	Static	Dynamic	Dynamic
<b>Self-timed</b>	Static	Static	Static	Dynamic
<b>Fully static</b>	Static	Static	Static	Static

Table 11 Extended scheduling classifications

If processor assignment is static, the communication also follows the same static pattern and is known beforehand. Therefore, memory assignment can be static. However, when processor assignment becomes dynamic, it also means the communication path between processors might change, enabling dynamic buffer assignment.

## 6.3 Static mapping

Static processor assignment is the most commonly used scheduling approach for embedded systems. Each task is assigned to one processing unit, which will execute it. The mapping supports static assignment and self-timed schedules.

The scheduling problem involves taking a task graph  $(V, E, d, P, O, I)$  (see Section 1.3) and a hardware platform consisting of a set of processors  $P$  and memories  $M$ .

The static processor assignment involves finding a map  $a_p : V \rightarrow P$ , where  $V$  is the set of tasks and  $P$  is the set of processors. The map describes to which processor each task is assigned. For the simulation platform, the map is defined in the system definition.

The assignment of FIFO buffers to the memory is decided automatically. The processor local memories and a shared memory are mathematically defined as set  $M = \{m_p \mid p \in P\} \cup \{m_{shared}\}$ . A map  $a_m : E \rightarrow M$  can be defined. For each edge  $(u, v) \in E$  of the task graph, the mapping of the FIFO buffer is decided by the following conditions:

- If  $a_p(u) = a_p(v)$  then  $a_m((u, v)) = m_{a_p(u)} = m_{a_p(v)}$ .  
If both tasks are scheduled on the same processor, the processor-local memory is used.
- If  $a_p(u) \neq a_p(v)$  then  $a_m((u, v)) = m_{shared}$ .  
If the tasks are scheduled on different processors, communication must happen through the shared memory.

## 6.4 Dynamic mapping

With dynamic processor assignment, tasks are assigned to the processing units at run-time. The memory assignment is performed statically, resulting in a “dynamic task assignment” schedule. In the implementation, the actual processor assignment is performed during run-time by the system controller. When the simulation platform is constructed at compile-time, the mapping algorithm has to set-up the task instances on the various processing units.

The system definition is extended to specify on which processing units a task may execute. The restriction is used to allow the DFE to be placed on a separate processing unit, while the inner receiver tasks may be dynamically scheduled on another set of processing units.

The map, as specified in 6.3, is extended to specify the set of VPU's a task may be scheduled on. Mathematically this is expressed as a map  $a_p : V \rightarrow \wp(P)$ , where  $V$  is the set of tasks and  $\wp(P)$  is the power set of processing units. For a valid mapping, we will need to ensure that each task may run on at least one processor. This can be expressed as  $(\forall v : v \in V : a_p(v) \neq \emptyset)$ . At runtime, the scheduler must decide which processing unit from the set is chosen.

To simplify the analysis, and implementation of the mapping algorithm and scheduler, the choice is made to assign each task to a pool of processing units. A pool is a group of processing units, with the restriction that each processing unit belongs to exactly one pool. Each task is assigned to a pool of processing units. The restriction can be expressed as  $(\forall u, v : u, v \in V : a_p(u) \cap a_p(v) \neq \emptyset \rightarrow a_p(u) = a_p(v))$ .

Since all tasks are assigned to exactly one pool, the total processing power of the pool must be sufficient to run the tasks which are assigned to it. Since a task may not run on two different pools, this simplifies the analysis.

The implementation can be simplified by introducing a “pool manager” to the system controller, as shown in Figure 48. When the global scheduler determines a task may be executed, it instructs the pool manager to choose an available processing unit to execute the task. The pool managers do not have to communicate with each other, since they manage their own pool of processing units.

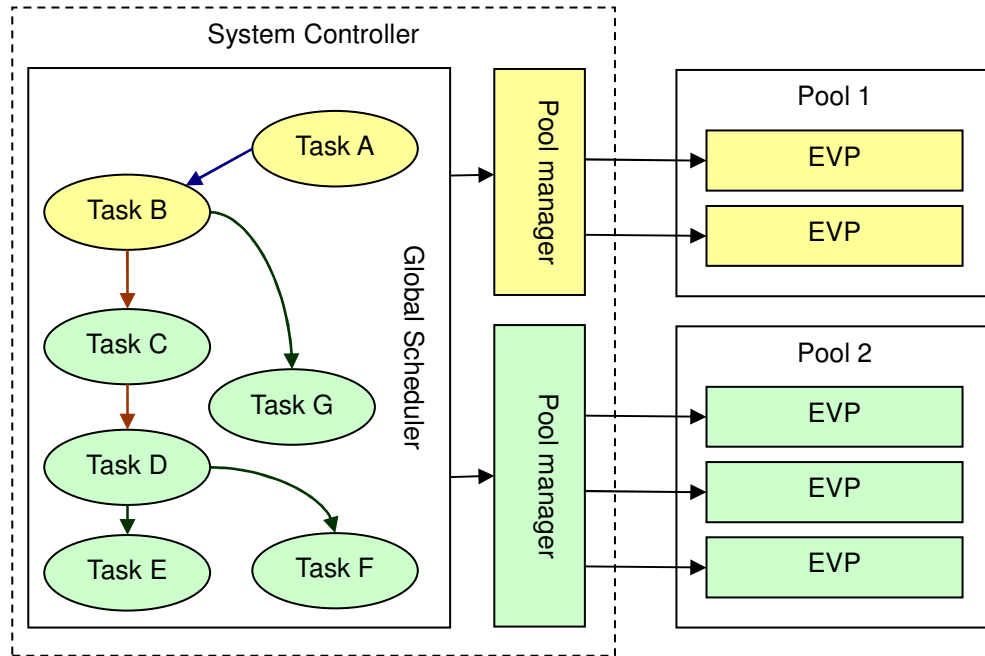


Figure 48 Using pool manager

The assignment of FIFO buffers to the memories is performed statically. The used memories can be decided automatically during compile-time. A set of memories  $\{m_p \mid p \in P\} \cup \{m_{shared}\} = M$ , consist of the processor local memories and a shared memory. A map  $a_m : E \rightarrow M$  can be defined. For each edge  $(u, v) \in E$ , the following conditions can hold:

- If  $\#a_p(u) = 1 \wedge \#a_p(v) = 1 \wedge a_p(u) = a_p(v)$  then  $a_m((u, v)) = m_{a_p(u)} = m_{a_p(v)}$ .  
If both tasks are guaranteed to be scheduled on the same processor at all times, the processor-local memory is used.
- Otherwise, then  $a_m((u, v)) = m_{shared}$ .

If the tasks can be assigned at run-time to different processors, communication must happen through the shared memory, as shown in Figure 49.

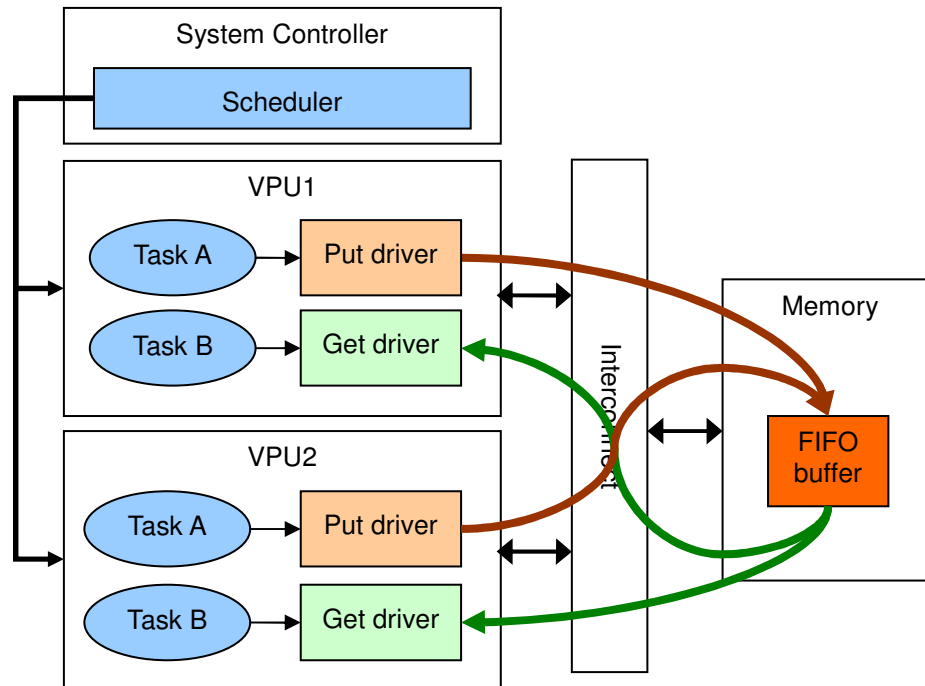


Figure 49 Dynamic Task Assignment

A major disadvantage is that tasks which are allowed to run on multiple processing units, must communicate using the shared memory. This happens, even if the tasks are assigned to the same processing unit during run-time. Therefore, system will spend more time on communication.

## 6.5 Communication-optimized mapping

The use of the dynamic mapping implies a more accesses on the memory subsystem, resulting in more energy consumed for communication. This also requires more processing units; since more cycles are spent on communicating.

By analyzing the task graphs for the antenna pre-processor and antenna combiner, the communication required can be calculated. In the LTE task graph given in Section 3.3, every token is a 32-bit quantity, resulting in the values shown in Table 12. For the static processor assignment as described in Section 3.3.1 is assumed. The “input” and “output” rows give the amount of data which must be received from and transmitted to tasks executing on other processors. The “internal” row gives the memory bandwidth between tasks mapped on the same processor. These tasks will communicate using the processor-local memory. The internal communicated information has to be stored in the processor-local memory and read back from it again. Therefore, it has to be counted twice in the totals.

	Antenna pre-processor		Antenna Combiner	
	kB/s		kB/s	
Input	124,616	6%	672,000	37%
Output	346,048	17%	201,600	11%
Internal	2 * 808,684	77%	2 * 470,400	52%
<b>Total</b>	<b>2,088,032</b>		<b>1,804,400</b>	

Table 12 Memory bandwidth for LTE in 20MHz and 4x2 MIMO mode

In Table 13 the total bandwidth requirements for the processor assignments is shown. For the static mapping, the mapping as described in Section 3.3.1 is used. The minimum achievable row lists the absolute minimum communication which is required for the inner receiver. The minimum is reached when all tasks are mapped onto a single processor and no communication between the antenna pre-processor and antenna combiner is required.

	kB/s	
Static mapping	1,814,928	100%
Dynamic mapping	5,990,464	330%
Minimum achievable	450,832	25%

Table 13 Memory bandwidth requirement for different mappings

For the dynamic mapping, where all tasks are allowed to run on any EVP, all data must be communicated using the shared memory. As a result, the memory bandwidth increases by a factor 4 for the antenna preprocessors and by factor 2 for the antenna combiner. With 2 antenna pre-processors and 1 antenna combiner, the total memory bandwidth requirement increases by more than a factor 3.

A fully dynamic scheduler also considers placement of the FIFO buffers during run-time. By ensuring most communication is performed using FIFO buffer located in processor-local memory, the overhead for communication is reduced. This placement forces both associated tasks to run on the same processing unit.

Due to lack of time, a more restrictive fully dynamic scheduler is chosen. Instead of identifying opportunities for memory bandwidth reduction at run-time, these are identified at compile-time and the tasks are clustered. To avoid changes to the existing system controller and task scheduler, the individual tasks in the cluster are grouped into a large task. Effectively, this forces the processor-local memory to be used for the communication within the cluster.

As the implemented task manager allows only a single instance of a task to execute at the same time, each task must be able to execute on a single processor. As a cluster is treated as a single task, the cluster must also be able to run on a single processor. Therefore, the cluster size is limited.

In Figure 50, the task graph for the antenna pre-processor is shown. The edges are annotated with the number of tokens send per invocation, indicating the most advantageous edges for clustering. The clustering is giving as an example and are picked by hand. The chosen clustering is not optimal.

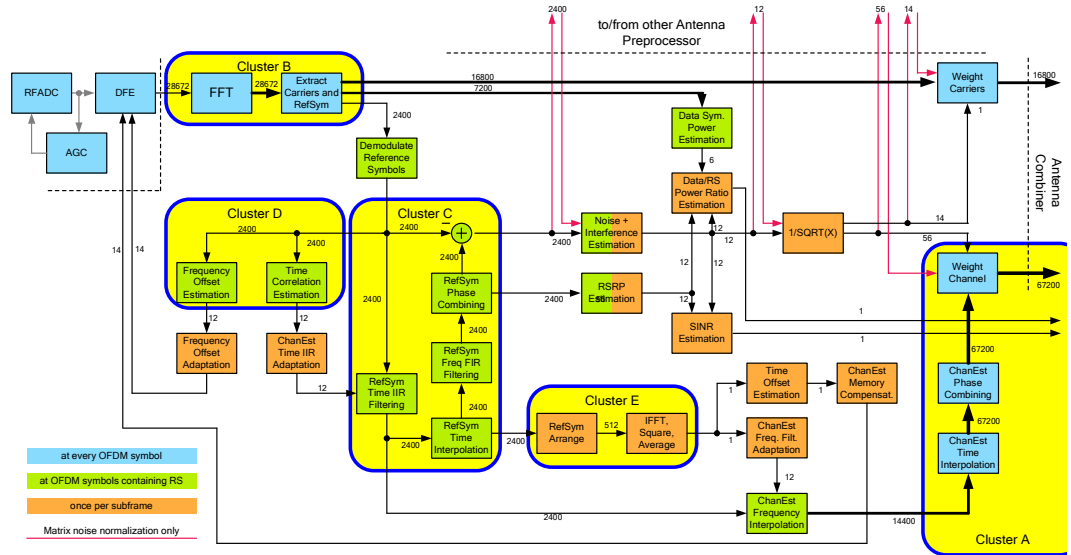


Figure 50 LTE Task Graph annotated with tokens per sub frame

In Table 14, the memory bandwidth savings of the proposed clusters are computed. The cluster A provides a significant saving of 54%, making it very worthwhile. Cluster B and C provide also. On the other hand, cluster D and E provides only minimal saving and do not seem worth the effort. It must be noted that, the clusters C and D use an optimization which is not obvious from the task graph: all connected tasks receive the same data from the “Demodulate Reference Symbols” task. Hence the communication can be reduced.

	Task	Communication saving (kB/s)	Cycles / Subframe
A	ChanEstPhaseComb	2 * 2 * 806,400 (54%)	112,896
	ChanEstTimeInt		
	WeightChannel		
	Precoding Matrix Mult.		
B	FFT	2 * 2 * 114,688 (7.7%)	55,720
	ExtractCarriers		
C	RefSymTimeIRFilt	2 * 2 * 48,000 (3.2%)	15,600
	RefSymTimeInt.		
	RefSymFreqFIRFilt		
	RefSymPhaseCombine		
	Addition		
D	FreqOffsetEst	2 * 2 * 9,600 (0.6%)	1,920
	TimeCorEst		
E	RefSymArrange	2 * 2 * 2,048 (0.1%)	5,005
	IFFTSqAvg		
<b>Total</b>		<b>3,922,944 (65%)</b>	

Table 14 Clustering effects

The clustering example shows that roughly 2/3<sup>rd</sup> of the bandwidth can be saved. As a result, the memory bandwidth requirements are nearly the same as with the static mapping, as shown in Table 15.

	kB/s	
Static mapping	1,814,928	100%
Dynamic mapping	5,990,464	330%
Dynamic with clustering	2,067,520	114%

Table 15 Memory bandwidth requirement for different mappings

In addition, since Cluster A requires 112,896 cycles per sub frame, it means the processor has to run at least at 113 MHz, without accounting for the overhead of communication.

A fully dynamic algorithm should be able to find more opportunities for reducing memory bandwidth by scheduling the task graph on-the-fly. The dynamic placement of buffers is left for future work.

# 7 Results and Discussion

This section will discuss the experiments which are conducted and the results obtained.

Conclusions for different bus models cannot be drawn, since the simulation failed to work. The generated database with results becomes corrupted and results cannot be extracted.

## 7.1 Experiment Set-up

For the experiments, the LTE application, as described in Section 3.3, will be mapped onto hardware. The LTE graph supports different modes of operation, which have to be explored.

The hardware platform, as shown in Figure 51, consisting of:

- A set of EVPs.  
The EVP is ST Ericssons digital signal processor optimized for cellular algorithms. The EVPs will be running at full speed of 312 MHz or in a low-power mode with frequency divider of  $1/3^{\text{rd}}$ , giving 104 MHz. The EVPs will be executing the task graph.
- Two sources: the RF/ADC with digital front-ends.
- One sink: the outer receiver.
- System controller.  
The system controller will perform global control by directing the EVPs to execute the task graph.
- Uniform Access Memory.  
The memory interface is 64-bit wide, runs at 312 MHz and has a latency of 48 cycles.

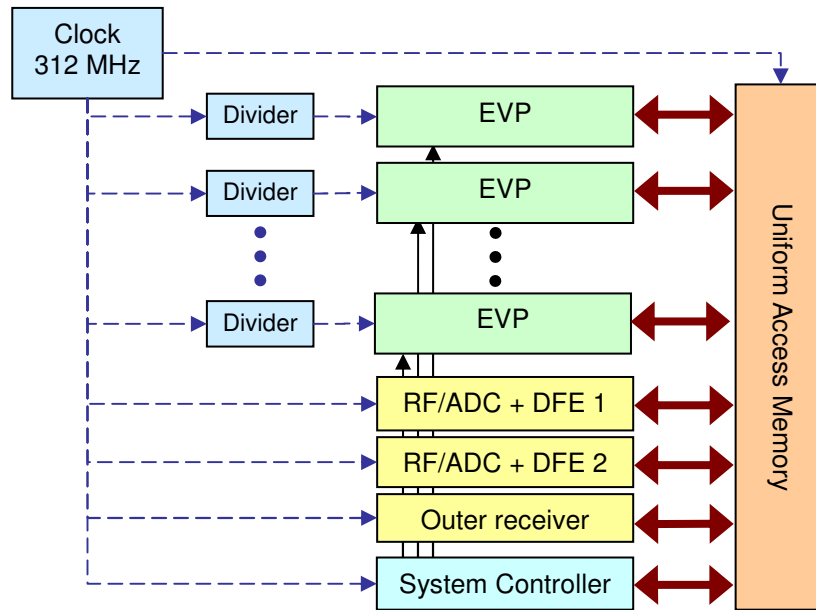


Figure 51 Experimentation platform

For energy consumption measurements only the EVPs and memory are considered. The other elements are not part of the inner receiver and are not analyzed. The energy consumption for the EVP is shown in Table 16. The memory will require 0.05 nJ per 64-bit transfer.

Operating mode	Operating Frequency	Energy per cycle (nJ / cycle)		Operating Power (mW)
		Operating	Idle	
Full speed	312 MHz	0.5	0	156
Low power	104 MHz	0.25	0	26

Table 16 EVP energy consumption

Each experiment will describe:

- Operation mode of the application.
- Number of EVPs used and their operation mode.
- The mapping method used.

Any other variations to the architecture will be described when the experiment is conducted.

The LTE task graph has a period of 1 ms. For self-timed schedulers, the task graph will enter a period regime after some startup time. The measured values are taken from the simulation between 40 and 50 ms. The time period contains 10 iterations of the task graph and smoothens any variations in processing.

## 7.2 Experiment 1: Simulation Validation

The first experiment aims to validate the results of the simulation and thereby show the simulator is functioning as expected. The following parameters are used:

- LTE task graph with bandwidth 20 MHz and 4x2 MIMO mode.
- 3 EVPs running at full speed are used.
- The application is statically mapped according to Section 3.3.1.

The energy consumption of the system controller, RF/ADCs and DFEs and outer receiver are not taken into account, since they are not part of the inner receiver. Only the memory bandwidth requirements and energy consumed are counted, as shown in Table 17. These values do not vary between different mappings and are not relevant.

	Memory load (kB/s)	Memory power (mW)
<b>System Controller</b>	None	0
<b>RF/ADC + DFE 1</b>	112,219	0.72
<b>RF/ADC + DFE 2</b>	112,219	0.72
<b>Outer receiver</b>	196,906	1.26

*Table 17 Fixed elements energy consumption*

The expected computation times for the antenna pre-processors and antenna combiner can be computed and are displayed in Table 18. In Table 6 (on page 37) and Table 7 (on page 38) the number of cycles required per sub frame is computed. As both antenna pre-processors execute exactly the same operations, their expected values are the same. It is known that a sub frame takes 1 ms and the EVPs run at 312 MHz. With this information the VPU load can be computed.

The memory load has been analyzed in Section 6.5 and is repeated here. Since communication of 64-bit value requires 1 cycle, the time spent by the EVP on communication is computed in "Memory VPU load". The total load of the VPU is the sum of the "VPU load" and "Memory VPU load" column.

The energy consumed by the VPU depends on the load and can be derived from the total VPU load and is shown in the "VPU power" column. The Memory power consumption depends on the number of bytes communicated and is shown in the "Memory power" column.

	VPU load	Memory load kB/s	Memory VPU load	VPU power	Memory power
<b>Pre-processor 1</b>	71.2%	459,632	18.9%	140	2.9
<b>Pre-processor 2</b>	71.2%	459,632	18.9%	140	3.0
<b>Combiner</b>	59.8%	853,125	35.0%	148	5.6
<b>Total</b>	<b>202%</b>	<b>1,772,389</b>	<b>73%</b>	<b>428</b>	<b>11.3</b>

*Table 18 Expected VPU and Memory usage and energy consumption*

The results from the simulation are shown in Table 19. The measured total VPU load is higher than the expected load by ~2.6% for the pre-processor and ~1.7% for the combiner. The memory values are a near perfect match, showing some inconsistency due to temporal effects.

	VPU load	Memory load kB/s	VPU power (mW)	Memory power (mW)
<b>EVP 1 (pre-processor 1)</b>	92.7%	459,820	145	2,9
<b>EVP 2 (pre-processor 2)</b>	92.7%	459,820	145	2,9
<b>EVP 3 (combiner)</b>	96.5%	852,727	151	5,5
<b>Total</b>	<b>281,9%</b>	<b>1,772,367</b>	<b>440</b>	<b>11,4</b>

Table 19 Measured VPU and Memory usage and energy consumption

The discrepancy of the VPU loads can be explained, since the expected values do not account for the memory latency incurred; only the actual memory transfers. The effects of latency are computed in Table 20.

	Transaction / sub frame	Cycles / sub frame	VPU load
<b>Pre-processor</b>	204	9,792	3.1%
<b>Combiner</b>	70	3,360	1.1%

Table 20 VPU load due to memory latency

By taking memory latency into account, the discrepancies decrease to 0.5%, which corresponds to the difference in memory measurement. This seems an acceptable value and may be caused by measurement errors. We show that the simulation performs as is expected and within a reasonable bound of error.

## 7.3 Experiment 2: Application Dynamism

In this experiment the dynamism of the application is explored. The LTE cellular standard specified different bandwidths and MIMO modes.

We will be using the mapping as described in Section 3.3.1; therefore the system will contain 3 EVPs.

### 7.3.1 Bandwidth scaling

The measurements show the scaling of the LTE algorithm with 3 EVPs running at full speed. The results are split into the antenna pre-processor in Table 21 and for the antenna combiner in Table 22. The combined energy requirements for the inner receiver are listed Table 23. The combined EVP load and memory bandwidth are shown in Figure 52, respectively Figure 53.

	VPU load	Memory load (KB/s)	VPU power (mW)	Memory power (mW)
20 MHz 4x2	92.7%	459.820	144,6	2,94
15 MHz 4x2	71.4%	345.102	111,4	2,21
10 MHz 4x2	49.2%	230.383	76,8	1,30
5 MHz 4x2	27.8%	115.664	43,4	0,74
3 MHz 4x2	17.0%	56.570	26,5	0,36
1.4 MHz 4x2	12.0%	28.758	18,7	0,18

Table 21 Scaling of pre-processor with LTE bandwidth

	VPU load	Memory load (KB/s)	VPU power (mW)	Memory power (mW)
20 MHz 4x2	96.5%	852.727	150.5	5.46
15 MHz 4x2	72.7%	639.844	113.4	4.10
10 MHz 4x2	49.0%	426.562	76.4	2.73
5 MHz 4x2	25.3%	213.281	39.5	1.36
3 MHz 4x2	13.0%	102.375	20.3	0.66
1.4 MHz 4x2	7.3%	51.188	11.4	0.33

Table 22 Scaling of antenna combiner with LTE bandwidth

	VPU power (mW)	Memory power (mW)	Total (mW)
20 MHz 4x2	440	11.34	451
15 MHz 4x2	336	8.52	345
10 MHz 4x2	230	5.33	235
5 MHz 4x2	126	2.84	129
3 MHz 4x2	73	1.38	75
1.4 MHz 4x2	49	0.69	50

Table 23 Combined energy consumption for LTE modes

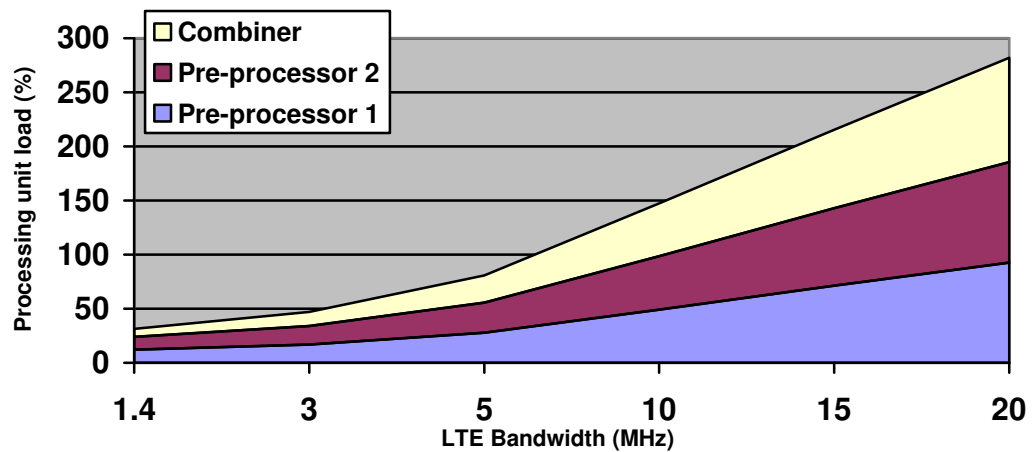


Figure 52 Combined processing power for LTE bandwidths

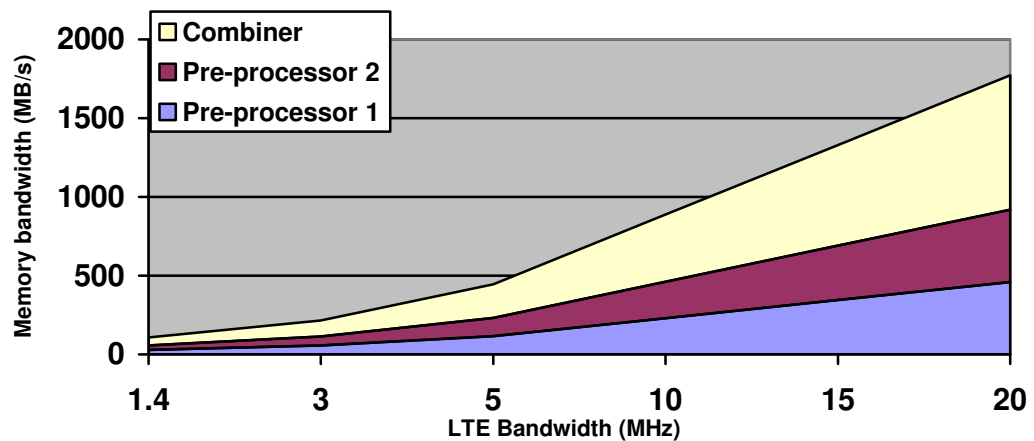


Figure 53 Combined memory bandwidth for LTE bandwidths

The VPU load and memory consumption scales nearly linearly with LTE bandwidths.

### 7.3.2 Energy optimization

The experiment has been conducted with 3 EVPs running at full speed. With the use of dynamic voltage and frequency scaling and different mappings, energy savings may be possible. The 1.4 and 5 MHz modes will be studied in detail, while results for overall scaling are also given. In the experiments only static mapping will be used. The following approaches have been explored:

- Reduce energy by running the EVPs, or a part of them, in a low power mode.
- Change the mapping by moving the tasks to another EVP.

The scaling for 1.4 MHz and 5 MHz is shown in Table 24 respectively Table 25. The system is tried with:

- 3 EVPs using the original Dresden mapping.

- 2 EVPs by combining the tasks of the antenna pre-processor onto a single EVP. This will reduce the overhead caused by the communication between the antenna pre-processors.
- 1 EVP by running the entire LTE algorithm on a single EVP. This reduces all communication.

1.4 MHz 4x2	Full power (mW)			Low power (mW)		
	VPU power	Memory power	Total	VPU power	Memory power	Total
3 EVPs	48.8	0,69	49,5 (100%)	18.6	0.69	19,3 (39%)
2 EVPs	44,1	0,67	44,8 (91%)	17.9	0.67	18,6 (38%)
1 EVP	34.3	0,17	34,5 (70%)	16.2	0.17	16,4 (33%)

Table 24 Energy consumption with different mappings and power modes for LTE at 1.4 MHz

5 MHz 4x2	Full power (mW)			Low power (mW)		
	VPU power	Memory power	Total	VPU power	Memory power	Total
3 EVPs	126.3	2.84	129,1 (100%)	50.1	2.84	52,9 (40%)
2 EVPs	120.9	2.77	123,7 (96%)	Cannot keep up		
1 EVP	95.0	0.68	95,7 (74%)			

Table 25 Energy consumption with different mappings and power modes for LTE at 5 MHz

The results show that solely using a low power mode reduces energy consumption by 60%. Therefore, we can conclude that implementing low power modes is very worthwhile.

A simple mapping change, where one EVP runs all the antenna pre-processor tasks provides little benefit and is not always possible. However, the change is relatively simple to implement, since the algorithms for both pre-processors are the same. As a result, no additional software has to be added to the antenna pre-processor. Merging all tasks onto a single processor provides an addition gain up to 30%, and up to 15% in a low power mode.

The improved energy consumptions which can be reached with minor changes in the mapping and EVP operating modes are listed in Table 26. It is shown that by applying dynamic voltage and frequency scaling and using different mappings, very significant energy savings are possible.

Figure 54 the power consumption is compared graphically. In Figure 55, the energy consumption per megabit is compared.

Mode	Mapping	Power	Saving
20 MHz 4x2	3 EVPs in full power mode	451.0	0%
15 MHz 4x2	3 EVPs in full power, precoding matrix multiplication on antenna pre-processor	314.9	8,6%
10 MHz 4x2	2 EVPs in full power for antenna pre-processor and 1 EVP in low-power mode for combiner	179.9	23,5%
5 MHz 4x2	3 EVPs in low power	52.9	59,0%
3 MHz 4x2	1 EVP in low power mode	25,6	65,7%
1.4 MHz 4x2	1 EVP in low-power mode	16,4	66,9%

Table 26 Improved energy consumption with LTE bandwidth scaling

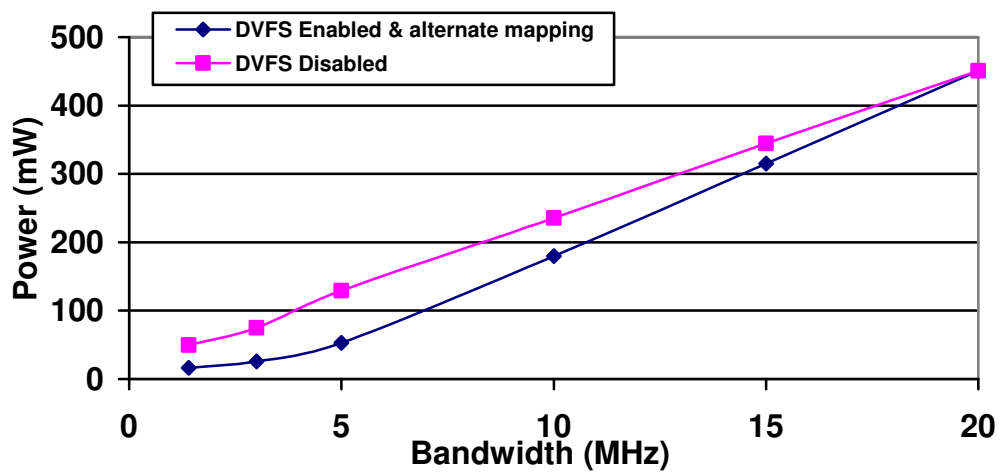


Figure 54 Energy consumption for different LTE bandwidths

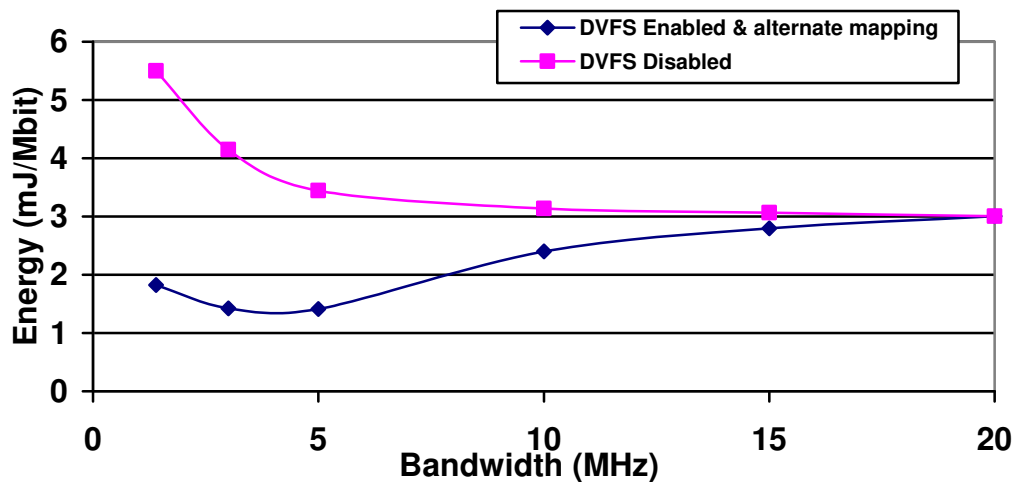


Figure 55 Energy per megabit for different LTE bandwidths

It has been shown that the power consumption can be significantly reduced, even on the existing hardware. The energy consumption per megabit can be improved and will improve at lower speeds, using voltage and frequency scaling. For 1.4 MHz LTE bandwidth

mode, the fixed overhead seems to be dominating and further improvement is no longer possible.

## 7.4 Experiment 3: Dynamic assignment with Clustering

In this experiment the overhead for dynamic assignment is analyzed. In Section 6.5 an improvement aimed to significantly reduce memory bandwidth is presented. These experiments analyze the added cost of using dynamic assignment. The following setup is used:

- The application is LTE at 20 MHz and 4x2 MIMO mode.
- 6 EVPs running at full speed.

In Table 27 the results of using the LTE task graph with dynamic mapping. The memory bandwidth requirement has increased significantly to nearly 6.5 GB/s compared to 1,7 GB/s required for static mapping. As a result, more time is spent by the EVP on communication and energy consumption increases significantly. The memory load is higher than is expected in Section 6.5.

	VPU load	Memory load (MB/s)	VPU power (mW)	Memory power (mW)
<b>EVP 1</b>	83.7%	1,076	131	6.89
<b>EVP 2</b>	83.5%	1,070	130	6.85
<b>EVP 3</b>	82.9%	1,076	129	6.89
<b>EVP 4</b>	83.5%	1,084	130	6.94
<b>EVP 5</b>	83.4%	1,080	130	6.91
<b>EVP 6</b>	84.1%	1,083	131	6.93
<b>Totals</b>	<b>501%</b>	<b>6,471</b>	<b>782</b>	<b>41.4</b>

*Table 27 Dynamic Mapping without clustering*

As the memory overhead has increased significantly, and therefore energy consumption also increased significantly. In order to improve the results, the excessive communication must be reduced. By clustering tasks, as shown in Figure 56, the usage of the shared memory is reduced and the processor-local memory is used instead. As a result, the energy consumption should be reduced. As can be seen in Table 28, an 10% improvement in energy consumption is attained.

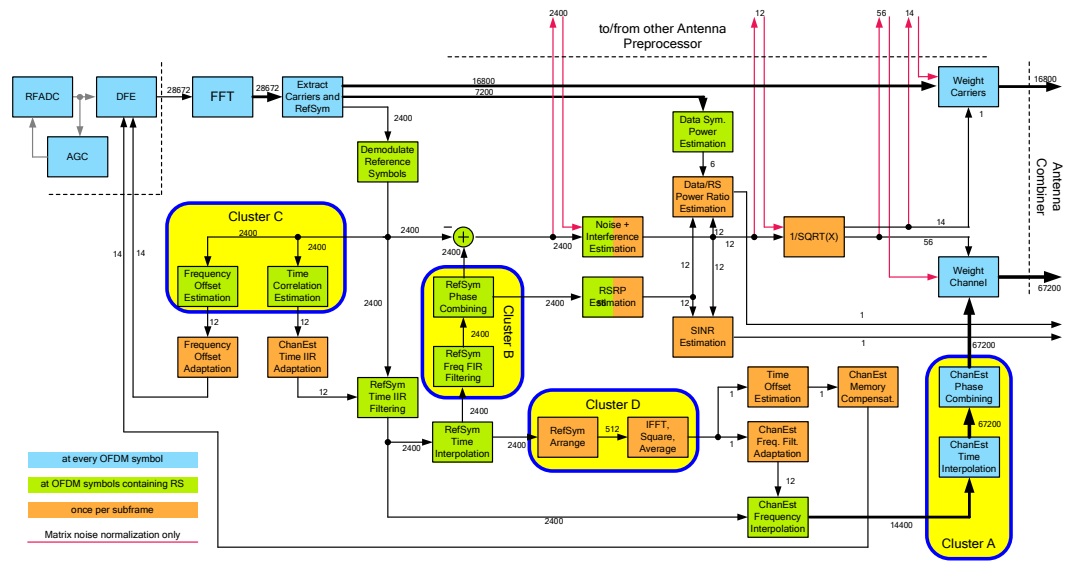


Figure 56 Initial task graph clustering

	VPU load	Memory load (MB/s)	VPU power (mW)	Memory power (mW)
<b>EVP 1</b>	74.5%	898	116,2	5,75
<b>EVP 2</b>	74.1%	882	115,6	5,65
<b>EVP 3</b>	76.1%	897	118,7	5,74
<b>EVP 4</b>	76.1%	903	118,7	5,78
<b>EVP 5</b>	74.5%	881	116,2	5,64
<b>EVP 6</b>	74.4%	877	116,1	5,61
<b>Total</b>	<b>450%</b>	<b>5,337</b>	<b>702</b>	<b>34,2</b>

Table 28 Dynamic Assignment with clustering

Since clustering seems a good solution, the more aggressive clustered, as illustrated in Section 6.5, is performed. As the shared memory is less used and less communication takes place between the processors, the energy consumption should decrease significantly. The results are presented in Table 29.

	VPU load	Memory load (MB/s)	VPU power (mW)	Memory power (mW)
<b>EVP 1</b>	55.5%	442	86,6	2,83
<b>EVP 2</b>	54.2%	444	84,6	2,84
<b>EVP 3</b>	53.4%	418	83,3	2,67
<b>EVP 4</b>	54.0%	431	84,2	2,76
<b>EVP 5</b>	55.6%	457	86,7	2,93
<b>EVP 6</b>	54.6%	446	85,2	2,85
<b>Total</b>	<b>327%</b>	<b>2,638</b>	<b>511</b>	<b>16,9</b>

Table 29 Dynamic Assignment with extensive clustering

The improvement with more extensive clustering is clear, since the energy consumption, as shown in Figure 57 is improving. By using clustering, the power consumption is 528 mW, which is a major improvement of the first 823 mW. Still, the result is 17% higher than the 451 mW required for static mapping. We can conclude that the dynamic scheduler should find more opportunities to save bandwidth to compete with static mapping.

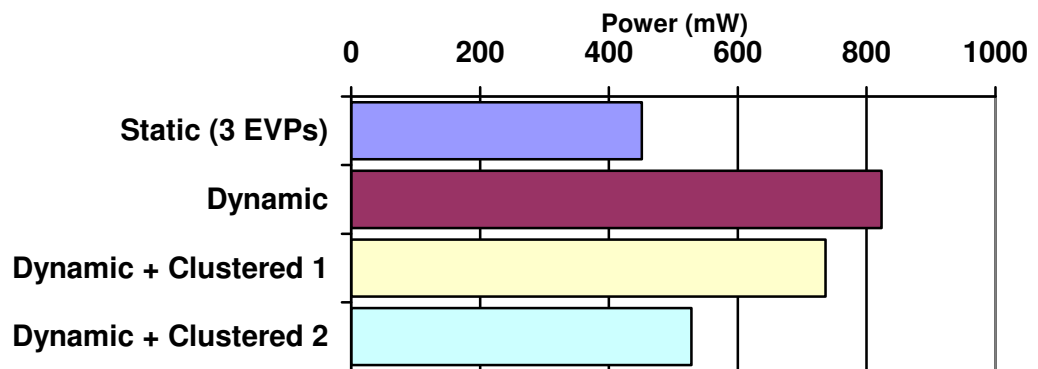


Figure 57 Dynamic assignment comparison

The achieved results are higher than expected from Table 15 (on page 67). The difference is similar from all experiments, indicating an oversight or miscalculation in earlier computations.

## 7.5 Experiment 4: LTE Multi Processor Scaling

In this experiment the scaling of the dynamic scheduling algorithm with different number of processors is analyzed.

- The application is LTE at 20 MHz and 4x2 MIMO mode.
- The dynamic mapping, as in Section 6.4, and dynamic mapping with clustering, as in Section 6.5 are used.

Since the amount of work to be done is fixed and the memory communication, by means of clustering, is also set a compile-time, it is expected that the total processor usage and memory usage will not vary with the number of processors and the totals will be equal.

## 7.5.1 Dynamic mapping

In Table 30, Table 31 and Table 32 the results from dynamic mapping with 6, 8 respectively 12 EVPs running at full speed are shown. It can be observed that the VPU load and memory load of each VPU gradually drop as more VPUs are added.

	VPU load	Memory load (MB/s)	VPU power (mW)	Memory power (mW)
<b>EVP 1</b>	83.7%	1,076	130.6	6.89
<b>EVP 2</b>	83.5%	1,070	130.3	6.85
<b>EVP 3</b>	82.9%	1,076	129.3	6.89
<b>EVP 4</b>	83.5%	1,084	130.3	6.94
<b>EVP 5</b>	83.4%	1,080	130.1	6.91
<b>EVP 6</b>	84.1%	1,083	131.2	6.93
<b>Total</b>	<b>501%</b>	<b>6,471</b>	<b>782</b>	<b>41.4</b>

Table 30 Dynamic mapping with 6 EVPs

	VPU load	Memory load (MB/s)	VPU power (mW)	Memory power (mW)
<b>EVP 1</b>	63.8%	817	99.5	5.23
<b>EVP 2</b>	62.5%	794	97.5	5.08
<b>EVP 3</b>	61.2%	818	95.5	5.24
<b>EVP 4</b>	63.5%	810	99.1	5.19
<b>EVP 5</b>	63.9%	813	99.7	5.20
<b>EVP 6</b>	62.5%	808	97.5	5.17
<b>EVP 7</b>	63.1%	798	98.4	5.11
<b>EVP 8</b>	61.6%	810	96.1	5.18
<b>Total</b>	<b>502%</b>	<b>6,470</b>	<b>783</b>	<b>41.4</b>

Table 31 Dynamic mapping with 8 EVPs

	VPU load	Memory load (MB/s)	VPU power (mW)	Memory power (mW)
<b>EVP 1</b>	41.7%	538	65,1	3,44
<b>EVP 2</b>	43.9%	550	68,5	3,52
<b>EVP 3</b>	41.5%	535	64,7	3,43
<b>EVP 4</b>	42.2%	541	65,8	3,46
<b>EVP 5</b>	41.3%	527	64,4	3,37
<b>EVP 6</b>	41.2%	545	64,3	3,49
<b>EVP 7</b>	42.5%	549	66,3	3,51
<b>EVP 8</b>	41.7%	530	65,1	3,39
<b>EVP 9</b>	41.7%	545	65,1	3,49
<b>EVP 10</b>	40.1%	532	62,6	3,40
<b>EVP 11</b>	41.1%	539	64,1	3,45
<b>EVP 12</b>	42.8%	540	66,8	3,45
<b>Total</b>	<b>502%</b>	<b>6,470</b>	<b>783</b>	<b>41,4</b>

Table 32 Dynamic mapping with 12 EVPs

From the results it can be concluded that the total required computing power, memory bandwidth and energy consumption remain the same. The total processor load is a result of the required computations, which remain the same. Only the amount of resources increases and allows spreading the workload. In addition, the memory bandwidth is also fixed, since the memory assignment is static. As power depends on the workload and memory traffic, the power also doesn't change between different mappings.

In Table 33 the results for dynamic mapping are shown and in Table 34 the results for dynamic mapping with clustering are shown. Regardless of the number of processors, the energy consumption remains the same, which is caused by the fixed costs for processing and communication.

DVFS mode	# EVPs	4	5	6	8	12	
<b>Full speed</b>	<b>VPU</b>	System cannot keep up			782	783	783
	<b>Memory</b>				41.4	41.4	41.4
	<b>Total</b>				823	824	824
<b>Low-power</b>	<b>VPU</b>	System cannot keep up					
	<b>Memory</b>						
	<b>Total</b>						

Table 33 Results for dynamic mapping

Due to excessive communication, at least 6 EVPs are required. The low power modes were not utilized. It is expected that for dynamic mapping at least 18 EVPs are required, since they run at 1/3<sup>rd</sup> of the clock frequency. It has not been possible to simulate this

amount of processors, since CoWare post-processing tool, which is used for analyzing the simulation results, crashes when observing the data.

## 7.5.2 Dynamic mapping with clustering

The excessive communication required for dynamic mapping has been observed in experiment 3 and the use of aggressive clustering shows greatly improved results. The results for processor scaling with this algorithm are shown in Table 34.

DVFS mode	# EVPs	4	5	6	8	12
Full speed	VPU	508	508	509	509	509
	Memory	16.9	16.9	16.9	16.9	16.9
	Total	525	525	526	526	526
Low-power	VPU	System cannot keep up. Too large clustered task for scheduler.				
	Memory					
	Total					

Table 34 Results for dynamic mapping with clustering

For dynamic mapping with clustering, one of the clusters has been chosen too large, not allowing single processor to execute it. It has been noted in Section 6.5 that a processor running at least 113 MHz is required. Since in low power mode the EVP runs at 104 MHz, the performance requirements for the method are not met.

By reducing the application to 10 MHz with 4x2 MIMO it suddenly becomes possible put the EVPs in low-power mode and reduce energy consumption, as shown in Table 35. Due to the availability of more resources, the energy consumption has been dropped significantly.

	VPU load	Memory load (MB/s)	VPU power (mW)	Memory power (mW)
EVP 1	76.6%	259	19,9	1,66
EVP 2	75.7%	273	19,7	1,74
EVP 3	75.9%	266	19,7	1,70
EVP 4	76.3%	262	19,8	1,68
EVP 5	76.0%	262	19,8	1,68
Total	381%	1,321	99	8,5

Table 35 Dynamic mapping with clustering of 10 MHz 4x2 on 5 EVPs

## 7.6 Discussion

The experiments have shown that the simulation is a fast method for exploring different architectures. The simulation platform supports various hardware architectures, software

models and mapping methods. It has been shown that dynamic voltage and frequency scaling provide the potential for large energy savings.

From the experiments it becomes obvious that the energy consumption seems to be of two major factors:

1. Processing power.  
Energy is consumed by the EVP. At full speed this is 0.5 nJ per cycle.
2. Communication power.  
Energy is consumed by the EVP and by the memory and interconnect. With 64-bit per word and the EVP at full speed, the energy cost is  $0.5 + 0.05 = 0.55$  nJ per 64-bit word.

As a result, communication is more expensive than computations in this model. Even though the energy consumed by the memory remains a relative small fraction of the total power consumption.

The result is that dynamic assignment suffers greatly from the increased communication overhead. In fact, it needs twice as much EVPs than with static mapping. By using clustering the memory requirements can be significantly reduced and performance improves to the point that only 4 EVPs are sufficient. Still, due to the higher memory bandwidth requirement, the dynamic mapping does not seem opportunistic in its current form.

The same conclusions hold when the energy cost for communication would be lower. As the execution of the algorithms will take the same amount of cycles, the only way to lower their cost is by reducing the operating frequency and voltage of the processors. Any communication will be overhead, which will consume time on the processing unit and cost additional energy.

What is needed, is a dynamic scheduler which can find ways to reduce memory bandwidth, compared to the static assignment. The reduced memory bandwidth not only reduces power, but also frees up the processor. As a result, the clock frequency of the processor can be reduced, saving even more energy.

In addition, the addition of more resources allows the devices to run at a lower speed, which will reduce power consumption.

# 8 Conclusions and Future Work

## 8.1 Conclusions

Analysis has been made of the new cellular standard with focus on the receiving and decoding parts. Software has been designed to explore the behaviour of the different applications implementing these cellular standards and the workload of these applications. The LTE-Advanced cellular standard is not yet been analyzed.

All parts of the receiver model have been simulated and a schedule process is implemented to control the processes.

The LTE algorithm has been shown to scale nearly linearly in processing power required. The only non-linear scaling elements are the FFT and IFFT; however their effects are not significant.

From the experiments it is found that the energy efficiency of the platform can be significantly enhanced by using dynamic voltage and frequency scaling. An energy reduction over 60% can be achieved for the inner receiver running on 3 EVPs.

By changing the processors assignment of the tasks, an additional savings of up to 15% becomes possible. When tasks are assigned to the same processor, the processor-local memory can be used for communication, which reduces the energy consumption.

So we can conclude that with these two techniques the total power consumption can be reduced with up to 67%. Minimal energy per bit is achieved at 3 and 5 MHz bandwidths.

Tasks with heavy communication must be assigned to a single processor. This is to save energy consumption.

It has been found that the dynamic processor assignment suffers from the increased communication using the shared memory. As a result, the current algorithm does not improve upon static mapping.

## 8.2 Future work

Investigate if dynamic processor assignment can reduce communication between processors over static processor assignment. If communication between processors is reduced, the processor loads and the energy consumption will decrease.

In future work also power-optimized scheduling algorithms should be analyzed that can control dynamic voltage and frequency scaling.

The task state is not modelled; this has to be taken into account in future version. Since it is not modelled, memory transfers for dynamic scheduling might be too optimistic.

## 9 References

- [1] 3GPP TR 25.814. Physical layer aspects for evolved Universal Terrestrial Radio Access (Release 7). V 7.1.0, 2006-09.
- [2] 3GPP TR 25.913. Requirements for Evolved UTRA and Evolved UTRAN (Release 7). V 7.3.0, 2006-03.
- [3] M. Bekooij, et. al. Dataflow Analysis for Real-Time Embedded Multiprocessor System Design. In: Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices, Chapter 4, pp. 81-108, Springer (Berlin), 2005.
- [4] C. van Berkel, F. Heinle, P. Meuwissen, K. Moerman and M. Weiss. Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices. EURASIP Journal on Applied Signal Processing. No. 2005:16, pp. 2613-2625, 2005.
- [5] C. van Berkel. Multi-Core for Mobile Phones. In DATE 2009 proceedings, pp. 1260-1266, 2009.
- [6] J. Berkmann, C. Carbonelli, F. Dietrich, C. Drewens and W. Xu. On 3G LTE Terminal Implementation – Standard, Algorithms, Complexities and Challenges. Wireless Communications and Mobile Computing Conference, 2008, pp 970-975, 2008.
- [7] G. Bilsen, M. Engels, R. Lauwereins and J. Peperstraete. Cyclo-Static Dataflow. IEEE trans. on signal processing, Vol. 44, No. 2, pp. 397-408, 1996.
- [8] G. Buttazzo. Hard Real-Time Computing Systems. Springer, ISBN 978-0-387-23137-2, 2004.
- [9] J. Cruz, A. Gonzalez, M. Valero, and N. Topham. Multiple-Banked Register File Architectures. Proceedings of the ISCA-27, pp. 316–325, 2000.
- [10] D. Culler and J. Singh. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, ISBN 1-55860-343-3, 1999.
- [11] E. Dahlman, S. Parkvall, J. Skold, P. Beming. 3G Evolution: HSPA and LTE for Mobile Broadband. Academic Press, ISBN 978-0-123-7253-32, 2007.
- [12] L. Goh, B. Veeravalli and S. Viswanathan. Design of Fast and Efficient Energy-Aware Gradient-Based Scheduling Algorithms for Heterogeneous Embedded Multiprocessor Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 1, 2009.
- [13] R. Gonzales, B. Gordon and M. Horowitz. Supply and Threshold Voltage Scaling for Low Power CMOS. IEEE Journal of Solid-state Circuits, Vol. 32, No. 8, pp. 1210-1216, 1997.
- [14] E. Lee and D. Messerschmitt. Synchronous data flow. IEEE proceedings, Vol. 75, no. 9, pp. 1235-1245, 1987.
- [15] E. Lee and D. Messerschmitt. Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE trans. on computers, Vol. C-36 No. 1:24-35, 1987.

- [16] E. Lee and D. Messerschmitt. Pipeline Interleaved Programmable DSP's: Synchronous Data Flow Programming. IEEE trans. on Acoustics, Speech and signal processing, Vol. ASSP-35, No. 9, pp. 1334-1345, 1987.
- [17] L. Miao, Y Qi, D. Hou and Y. Dai. Energy-Aware Scheduling Tasks on Chip Multiprocessors. IEEE Conference on Natural Computation, Vol. 4, pp. 319-323, 2007.
- [18] O. Moreira, J. Mol, M. Bekooij and J. van Meerbergen. Multiprocessor Resource Allocation for Hard-real-time Streaming with a Dynamic Job-mix. IEEE Proceedings of Real Time and Embedded Technology and Applications, 2005.
- [19] O. Moreira, J. Mol and M. Bekooij. Online Resource Management in a Multiprocessor with a Network-on-Chip. ACM, 2007.
- [20] O. Moreira and M. Bekooij. Self-Timed Scheduling Analysis for Real-Time Applications. EURASIP Journal on Advances in Signal Processing, Vol. 2007, Article ID 83710, 2007.
- [21] K. Nii, et al. A 45-nm Single-port and Dual-port SRAM family with Robust Read/Write Stabilizing Circuitry under DVFS Environment. IEEE Symposium on VLSI Circuits 2008, pp. 212-213, 2008.
- [22] R. Nuessgen. LTE Algorithm Architecture. ST Ericsson internal, DOC-104090, sept. 2008.
- [23] S. Sesia, I Toufik and M. Baker, Eds. LTE – The UMTS Long Term Evolution. Wiley, ISBN 978-0-470-6916-0, 2009
- [24] S. Sriram and S. Bhattacharyya. Embedded multiprocessors: scheduling and synchronization. Marcel Dekker, ISBN 0-8247-9318-8, 2000.
- [25] P. Wouters, M. Engels, R. Lauwereins, J. Peperstraete. Cyclo-dynamic dataflow. ESAT/ACCA/95/2, 1996.
- [26] Y. You, C. Lee, J. Lee and W. Shih. Real-Time Task Scheduling for Dynamically Variable Voltage Processors. IEEE workshop on Power Management for Real-Time and Embedded Systems, May 2001, pp. 5-10, 2001.

# 10 Acronyms and Terms

<b>3G</b>	Third Generation	<b>HSUPA</b>	High-Speed Uplink Packet Access
<b>3GPP</b>	Third Generation Partnership Project	<b>IDE</b>	Integrated Development Environment
<b>API</b>	Application Programming Interface	<b>LTE</b>	Long Term Evolution. A cellular standard.
<b>Arbiter</b>	A circuit dedicated to priority control of concurrent operations.	<b>MAC</b>	Media Access Controller
<b>CSDF</b>	Cyclo-Static Data Flow	<b>MIMO</b>	Multiple Input Multiple Output
<b>DFE</b>	Digital Front End	<b>OFDM</b>	Orthogonal Frequency Division Multiplexing
<b>DSP</b>	Digital Signal Processor	<b>QAM</b>	Quadrature Amplitude Modulation
<b>DVFS</b>	Dynamic Voltage and Frequency Scaling	<b>RF</b>	Radio Frequency
<b>EDGE</b>	Enhanced data rate for Global (or GSM) Evolution	<b>RS</b>	Reference Symbols
<b>EVP</b>	Embedded Vector Processor	<b>SDF</b>	Synchronous Data Flow
<b>FIFO</b>	First In, First Out	<b>SDR</b>	Software Defined Radio
<b>GPRS</b>	General Packet Radio Services	<b>UMTS</b>	Universal Mobile Telecommunications System.
<b>GSM</b>	Global System for Mobile communication	<b>VPU</b>	Virtual Processing Unit
<b>GUI</b>	Graphical User Interface		
<b>HSDPA</b>	High-Speed Downlink Packet Access		